

AI Assisted Workflows for Computational Electromagnetics and Antenna Design

Oameed Noakoasteen
PhD Defense Presentation
Version 05, Last Updated 2026.08

View this presentation on [Internet Archive](#)

View the manuscript on [UNM Digital Repository](#)

View the [GitHub Repository](#)

Outline

1. Research Objective:

- What is Artificial Intelligence?
- What is Machine Learning ?
- AI Assisted Workflows

2. Deep Learning:

- Feed Forward Networks (FFNs)
- Convolutional Layers
- Recurrent Layers
- Deep Feed Forward Networks
- Frameworks
- Experiment Setup in TensorFlow 2

3. Experiment-I:

Emulation of the FDTD using RNNs

4. Experiment-II:

Emulation of the FDTD using Transformers

5. Experiment-III:

Emulation of the FDTD using GNNs

6. Experiment-IV:

Device Modeling using GANs

7. Future Work

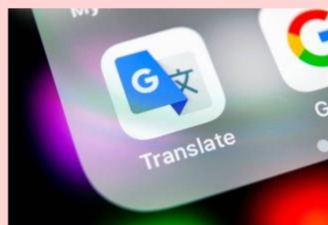
Research Objective: What is Artificial Intelligence?

Natural Language Processing (NLP)

Speech Recognition is the task of transforming spoken sound to text; and **Text-to-Speech Synthesis** is the task of transforming text to audible sound. Digital assistants interact with humans by listening to voice commands, executing some tasks (fetching news updates, placing purchase orders, setting timers, etc.) and then communicating the results in the form of human speech.



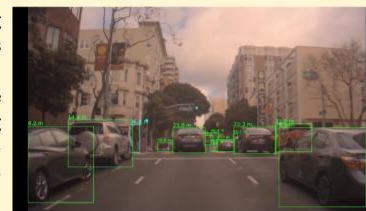
Machine Translation is the task of transforming text in one language into another. A translation system translates text from more than one hundred languages.



Computer Vision (CV)

Object Detection is the task of localizing (determining pixel locations using a bounding box) and classifying objects in an image.

An autonomous driving system finds the surrounding vehicles by first localizing and classifying objects in images captured by a front camera and then computing a distance estimate for each.



Pose Estimation is the task of detecting the position and orientation of objects or humans in a scene.

A robotic arm predicts 3D position and orientation of rigid objects for the purpose of manipulating them (picking, moving, and placing) in interactions with human collaborators.

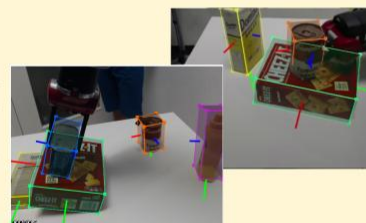


Image Synthesis is the task of generating a modified version of an existing image or generating an entirely new one. A facial expression generation system generates various emotional expressions from a neutral face while preserving its identity.

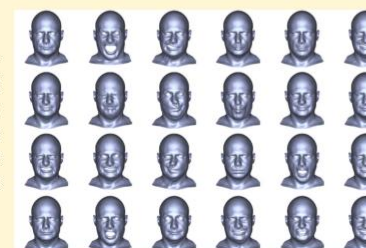


Image Transcription is the task of observing unstructured visual representation of some type of data and then transcribing the information into discrete textual form.

An image captioning system observes the scene depicted in the figure and provides a textual statement about its content.



a woman is throwing a frisbee in a park.

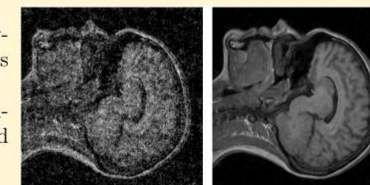
Image Super-Resolution is the task of generating a higher resolution version of the original image.

An image enhancement system enables fast acquisition of high-resolution MRI images by producing high-resolution reconstructions from low resolution acquisitions.



Image Denoising is the task of recovering the original image signal from its corrupted form.

An image restoration system reconstructs MRI scans from under-sampled inputs.



- The question “Can Machines Think?” was answered by Alan Turing with a thought experiment that became known as the **Turing Test**. In his proposed experiment, a computer program is intelligent if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Turing’s proposed test was an attempt at measuring intelligence by performance on some kind of open-ended behavioral task, rather than by philosophical speculation.

- Artificial Intelligence (AI) refers to computer programs that process linguistic (textual and auditory) and visual information to perceive and interact with the world like humans.

Research Objective:

What is Machine Learning?

Training: Two Guiding Principles

- Machine Learning (ML) refers to computer programs that accomplish their tasks by **learning** from examples.
- A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .
The **experience** that is available to the program is represented by the **dataset** which itself is a collection of **examples**. An example consists of a pair of **datapoint** and its corresponding **label/target**. A datapoint, is a collection of **features** that have been measured from some object or event.
For **classification** tasks the output is discrete-valued while for **regression** tasks the output is continuous-valued. Therefore, the **performance measure** is task specific. For classification tasks the **accuracy** (i.e., the proportion of examples for which the program produces the correct outputs) is used as the performance measure while for regression tasks the **average error** is used.
- ML is about creating value from data and its recent advancements (i.e., the development of Deep Learning) have come from the pursuit of artificial intelligence.
AI can't be achieved with explicit programming because it is infeasible to manually formalize vast amounts of information in a form that computers can use. In contrast, ML offers the ability to learn a task by experiencing a collection of examples.

The ML model, denoted by $f(\cdot, \mathbf{w})$, must perform its task in the **inference phase**; It will process examples that are considered to have been drawn from the data-generating distribution p_{data} , i.e., $(\mathbf{x}, y) \sim p_{\text{data}}$.
The ML model is constructed in the **training phase** using a collection of available examples, i.e., the **training set**, that are considered to have been drawn from the same p_{data} , i.e., $(\mathbf{x}^{\text{train}}, y^{\text{train}}) \sim p_{\text{data}}$. There are two guiding principles for constructing the model using the training set:

Empirical Risk Minimization

The optimal model, $f(\cdot, \mathbf{w}^*)$, is one that performs its task in the inference phase with minimal risk. The risk (also referred to as the **generalization error**) is defined as $R(\mathbf{w}) \equiv E_{(\mathbf{x}, y) \sim p_{\text{data}}} [L(f(\mathbf{x}, \mathbf{w}), y)]$. In other words, risk is the expected value of the **error** (i.e., the output of the **loss function** L) on examples that are drawn from p_{data} . Therefore, $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} (E_{(\mathbf{x}, y) \sim p_{\text{data}}} [L(f(\mathbf{x}, \mathbf{w}), y)])$.

However, since it is impossible to access all examples $(\mathbf{x}, y) \sim p_{\text{data}}$, the risk must be minimized indirectly. This is attempted by minimizing the empirical risk on the training set, hoping that doing so will also reduce the risk. The empirical risk is defined as $R_{\text{emp}}(\mathbf{w}) \equiv \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}, \mathbf{w}), y_i)$. In other words, the empirical risk is the average error for the examples of the training set. According to the Law of Large Numbers, $R_{\text{emp}}(\mathbf{w})$ will represent the $R(\mathbf{w})$ for a large enough training set. Therefore, $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} (\frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}, \mathbf{w}), y_i))$.

Likelihood Maximization

The optimal model, $f(\cdot, \mathbf{w}^*)$, is one that maximizes the likelihood of the weights, or equivalently, minimizes the **negative-log-of-the-likelihood** of the weights over the entire training set. Therefore, $\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} (-\log [p(\mathbf{y}|\mathbf{X}, \mathbf{w})])$.

Generalization

Once training is finished, the model enters the inference phase in which it will process previously unseen examples. If, after training, the gap between the average error computed on the training set and the average error computed on the **test set** (representing the generalization error) is minimal, it is said that the model **generalizes well**. Otherwise, the model **overfits** the training set. The model's generalization ability can be improved through **regularization**, which is any modification to the learning algorithm that is intended to reduce the gap between the average errors computed on training and test sets. The modified guiding principles are referred to as the **Structural Risk Minimization** and the **Posterior Maximization** respectively.

Research Objective: AI Assisted Workflows

Motivation

This line of inquiry is motivated by the fact that the large volumes of data (that we can easily store and manipulate these days) can be a rich resource for scientific discovery and development of technology if we can **extract valuable information** from them; And when it comes to deriving valuable information from data, **ML** emerges as the key solution.

Methodology

Devise and conduct experiments that aim to answer three fundamental questions:

- (1) Is it possible to obtain predictive models from the available simulation and measurement data?
- (2) What are the domain-specific machine learning algorithms required to convert various datasets to modeling knowledge?
- (3) Once the modeling knowledge has been learned, how to seamlessly incorporate it into a data-driven predictive environment?

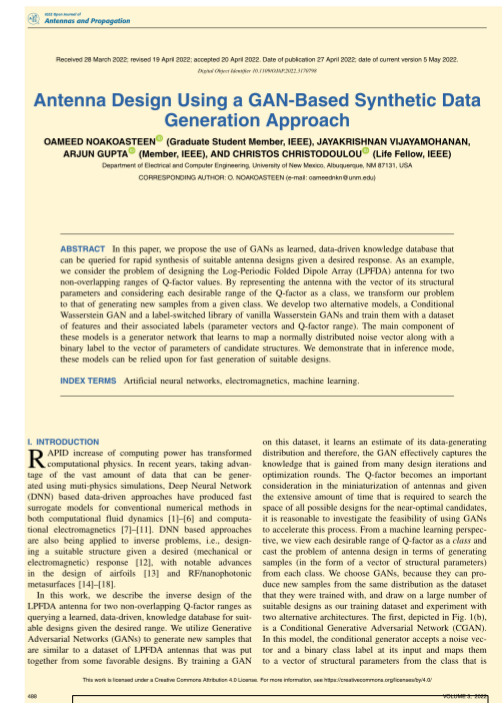
As will become clear, these experiments **recast engineering problems** as ML/AI tasks.

Thrust-I: Computational Electromagnetics



doi: [10.1109/OJAP.2020.3013830](https://doi.org/10.1109/OJAP.2020.3013830)

Thrust-II: Device Design & Development



doi: [10.1109/OJAP.2022.3170798](https://doi.org/10.1109/OJAP.2022.3170798)

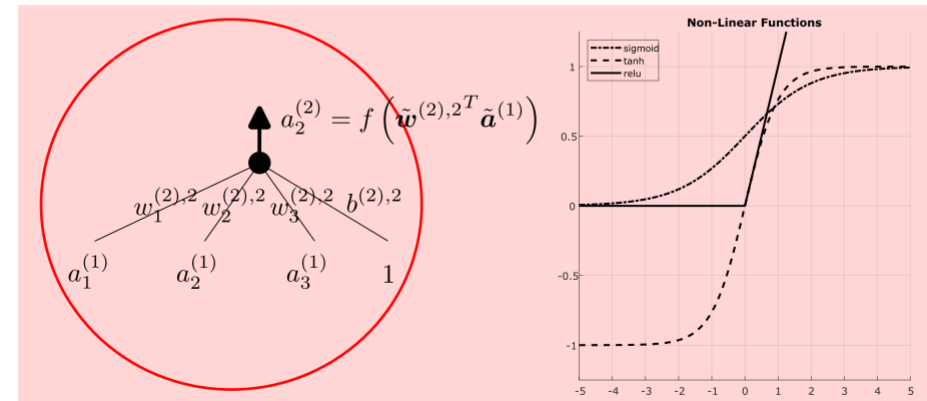
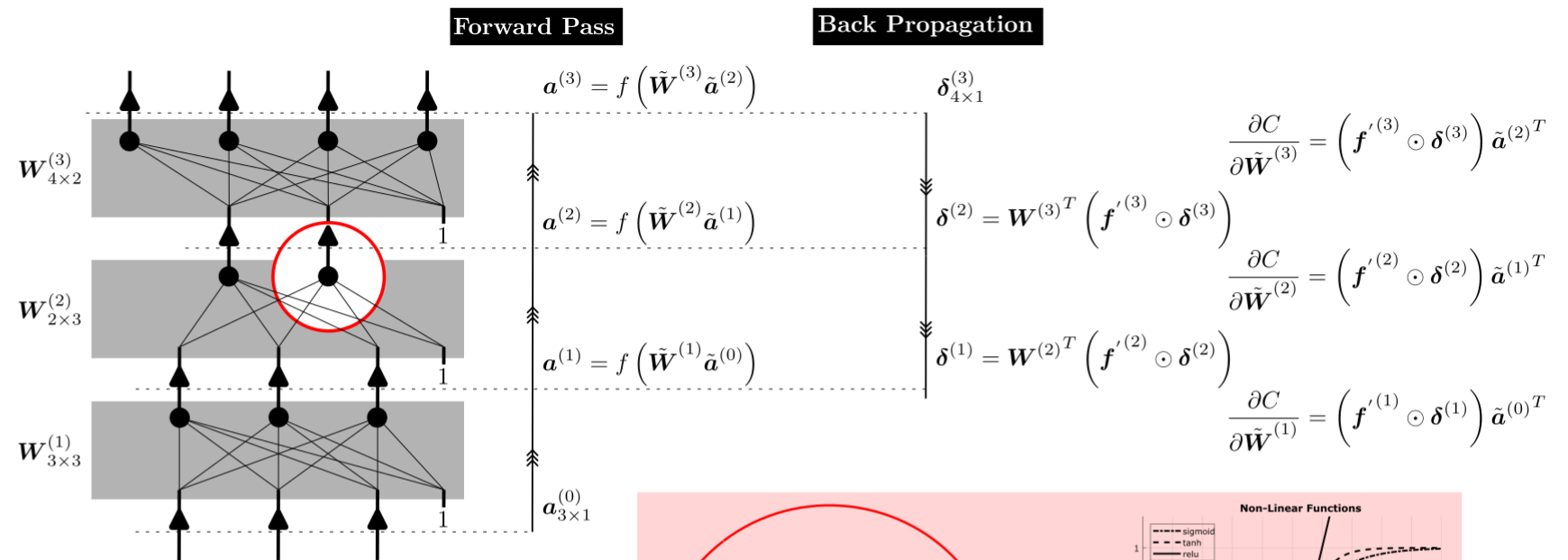
Deep Learning: Feed Forward Networks (FFNs)

■ **Artificial Neural Networks (ANNs)** are a category of ML models. They are computational models that are composed of multiple processing layers intended to learn representations of data at multiple levels of abstraction.

■ Each **layer** consists of several **units** (also referred to as **neurons**) that independently process that layer's input vector. A unit produces its output by multiplying each element of the input vector by a dedicated **weight**, summing them together along with a **bias** and, finally, applying a **non-linear function** to the total sum. Since each unit processes all the elements of the input, this type of layer is referred to as a **Fully-Connected** layer.

■ The output of a layer, which is the vector containing the outputs from all the units in that layer, is computed as the multiplication of the **expanded weight matrix** and the **expanded input vector**. The output of each layer becomes the input to the next layer and the computations continue.

■ In the training phase, the learning algorithm will need to compute the gradient of the cost w.r.t the weights and biases. For this purpose, the δ vector at the input of each layer is computed as the multiplication of the weight matrix with the scaled δ vector at the output of that layer.



Deep Learning: Convolutional Layers

■ A convolutional layer is a layer which is specialized for processing inputs that are in the form of a grid of values.

■ This type of layer can learn a single model for processing local neighborhoods of all the positions across the input grid.

This capability results from the layer's **sparse interactions** with its inputs and its output's **equivariance to translations**. The first property is achieved by restricting the interactions of the layer's set of weights (which are much smaller in size compared to the input) to only a local region of the input. The second property is achieved by **weight sharing** between different local regions.

■ Each element of the output is produced by sliding the set of weights, i.e., the **filter**, across the input. For each position, the weights in each channel of the filter are multiplied by their corresponding elements in the input (i.e., same channel and same position in the input) and the results are summed together. The sums from all the channels are added together along with a bias. Finally, a non-linear function is applied to the total sum.

$$W_{\text{out}} = \frac{W_{\text{in}} + 2P_w - D_w(F_w - 1) - 1}{S_w} + 1$$

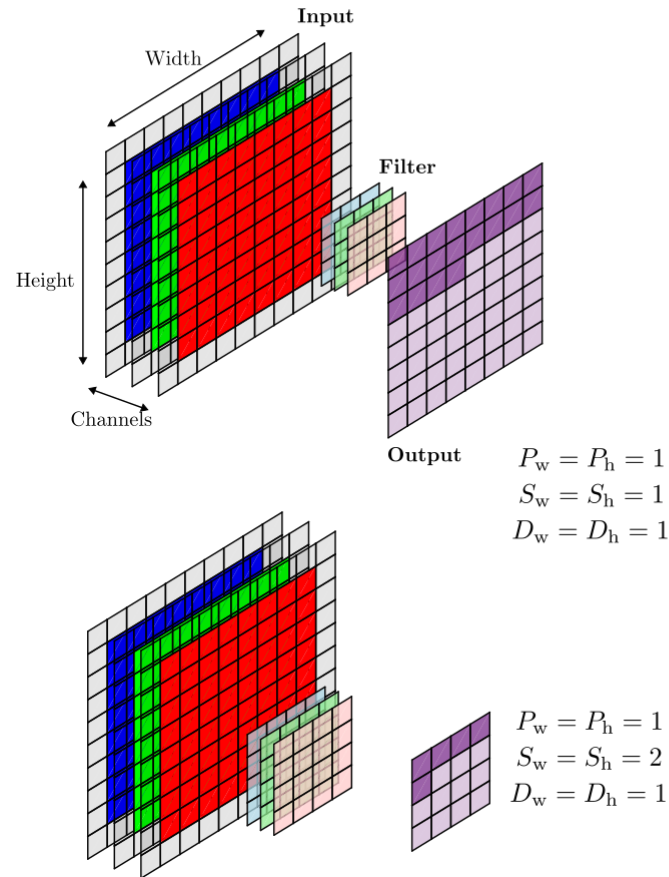
$$H_{\text{out}} = \frac{H_{\text{in}} + 2P_h - D_h(F_h - 1) - 1}{S_h} + 1$$

Padding refers to insertion of zeros to both sides (beginning and end) of the input in each dimension.

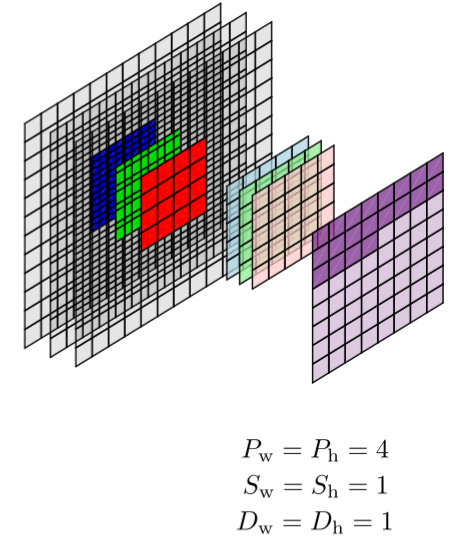
Stride refers to the distance between two consecutive positions of the filter.

Dilation refers to the inflation of the filter by inserting spaces between filter elements; This provides a convenient way to increase the receptive field (i.e., the local region of interaction) of the filter without increasing its size.

Same/Valid Convolution



Transpose Convolution



Deep Learning: Recurrent Layers

- A recurrent layer is a layer which is specialized for processing inputs that are in the form of sequences.

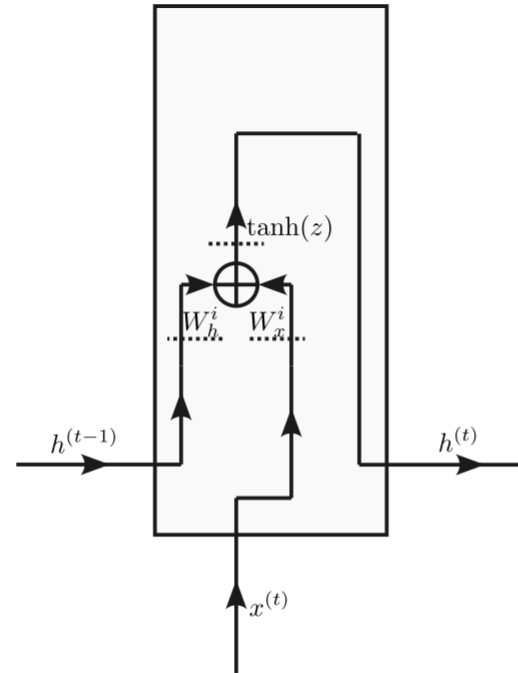
- This type of layer can learn a single model for processing all the positions of an input sequence. This capability results from **weight sharing** across all the positions of the input sequence which is achieved through **recurrent updates**.

A recurrent update refers to the application of the same forward propagation operation (i.e., multiplication by the same set of weights) to produce the current position in the output from the current input and the previous position in the output.

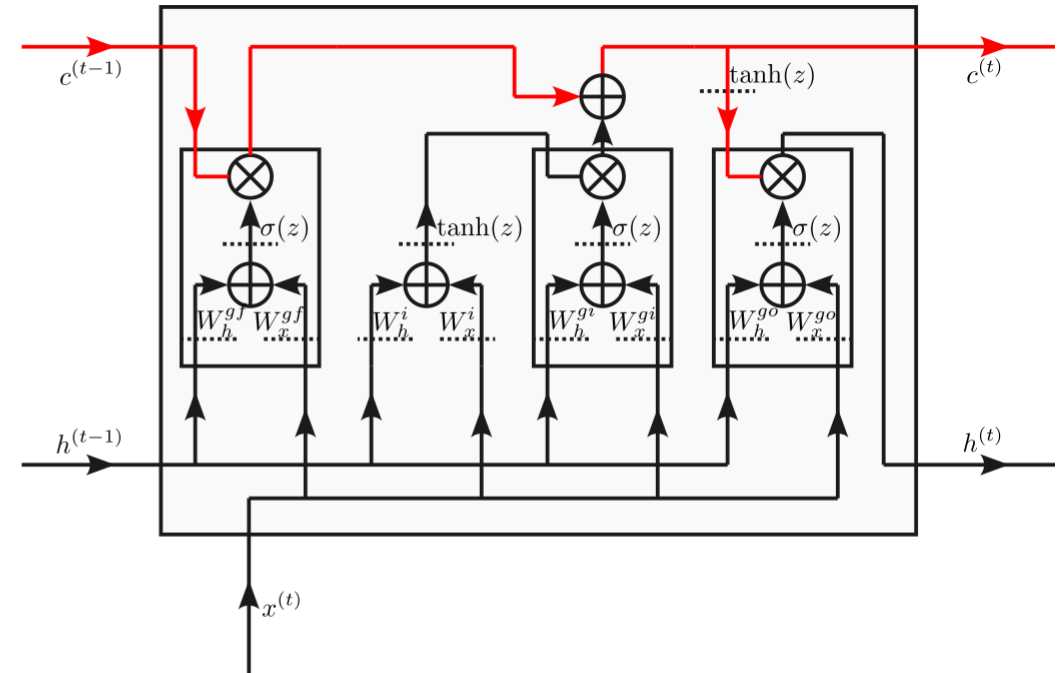
- The LSTM is the most suitable architecture for learning **long-term dependencies**. It comprises a **Constant Error Carousel** (CEC) and three gates, namely, the **forget gate**, the **input gate** and the **output gate**. The CEC acts as a memory mechanism which, selectively, holds the value from the previous input.

As such, the CEC provides an unattenuated/unmagnified path for the gradients of the cost as they propagate backwards through time and, therefore, helps to mitigate the **vanishing/exploding gradients** problem in long sequences.

Vanilla RNN



LSTM



Deep Learning: Deep Feed Forward Networks

- The **Universal Approximation Theorem** (UAT) states that a FFN with a linear output layer and at least one hidden layer which has any non-linear activation function (i.e., either a ReLU or a 'squashing' function such as the sigmoid) can approximate any Borel-measurable function (i.e., a continuous function defined on a closed and bounded subset of $\mathbb{R}^n$) with any desired non-zero amount of error, provided that the FFN is given enough hidden units.
- While the UAT guarantees the existence of the function that would generalize to unseen examples, it provides no guarantees that the learning algorithm will be able to learn that function. Specifically, learning can fail either due to the optimization algorithm's inability to find the set of parameters that corresponds to the desired function or due to overfitting.
- While the UAT guarantees the existence of a FFN large enough to achieve any desired degree of accuracy, it provides no indication as to how large it should be.
- In summary, a FFN with a single hidden layer is sufficient to represent any function, but the layer may fail to learn at all or to generalize and, also, it may be infeasibly large.
- FFNs are structured as deep stacks of layers, not just to alleviate the problems that the shallow networks present, but also to arrange for the sought-after function to be constructed by composition of several simpler ones.
- Training deep FFNs is fraught with difficulties. A major obstacle for the learning algorithm is the failure of the back-propagation procedure. The main failure mechanisms are the **vanishing/exploding gradients** and the **shattered gradients**.

Vanishing/Exploding Gradients

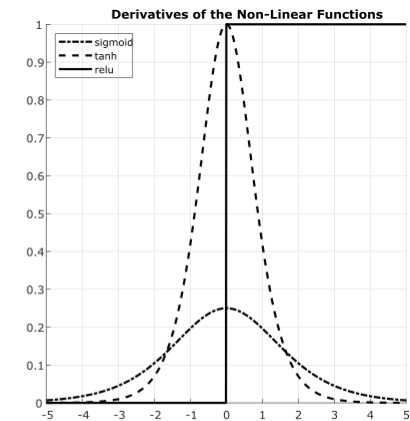
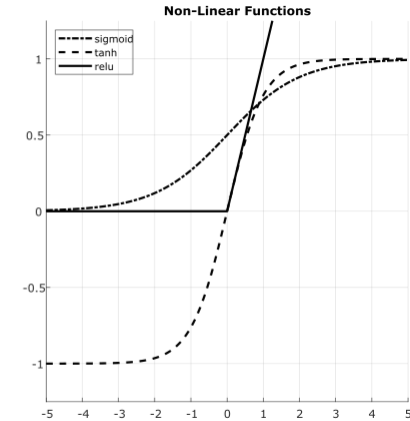
When the gradients of the cost w.r.t. the weights and biases of the l^{th} layer, i.e., $\frac{\partial C}{\partial \mathbf{W}^{(l)}}$, become so small that they can't effect meaningful change in that layer's parameters, the learning algorithm is said to suffer from **vanishing gradients**. Likewise, when $\frac{\partial C}{\partial \mathbf{W}^{(l)}}$ become so large that they cause unstable change, the learning algorithm is said to suffer from **exploding gradients**.

■ Non-saturating Non-linear Functions: ReLU

In computation of δ for deeper (i.e., lower) layers, the derivative of the non-linear function from all the previous layers are multiplied together. Since for the ReLU $f'(z \geq 0) = 1$, these multiplications will not attenuate the elements of the δ .

While $f'(z < 0) = 0$ does contribute to zero elements in δ and, consequently, in $\frac{\partial C}{\partial \mathbf{W}}$, the back-propagation procedure will work as long as the gradients can propagate along some paths; In other words, as long as $z \geq 0$ for some of the units in each layer.

In contrast to the ReLU, both the hyperbolic tangent and the sigmoid functions cause significant attenuation in elements of the δ in deeper layers even when their inputs are from a small interval centered at zero (i.e., are not from the saturated regions).



Deep Learning:

Deep Feed Forward Networks

■ Weight Initialization

Proper weight initialization ensures that, at the start of training, the magnitudes of $a^{(l)}$ and $\delta^{(l)}$ will not attenuate to zero or grow very large. With proper initialization, in the forward pass, starting from an input with zero mean and unit variance, the variance of the elements of the output of each layer remains the same as that for the layer below it. Likewise, in the back-propagation, the variance of the elements of the δ at the output of each layer remains the same as that for the layer above it.

He Initialization

For layers with ReLU activation, the weights can be drawn from either a normal distribution $w^{(l)} \sim \mathcal{N}(0, \frac{2}{O_{l-1}})$, in which O_{l-1} denotes the size of the input to the l^{th} layer, or a uniform distribution $w^{(l)} \sim U[-\sqrt{\frac{6}{O_{l-1}}}, \sqrt{\frac{6}{O_{l-1}}}]$.

Glorot Initialization

For layers with hyperbolic tangent activation, the weights can be drawn from either a normal distribution $w^{(l)} \sim \mathcal{N}(0, \frac{2}{O_l + O_{l-1}})$ or a uniform distribution $w^{(l)} \sim U[-\sqrt{\frac{6}{O_l + O_{l-1}}}, \sqrt{\frac{6}{O_l + O_{l-1}}}]$.

Shattered Gradients

When the gradients of the cost w.r.t the weights and biases of the l^{th} layer, i.e., $\frac{\partial C}{\partial \mathbf{w}^{(l)}}$, exhibit unexpected behavior, the learning algorithm is said to suffer from **shattered gradients**. This effect arises from the sequential dependence of layer outputs and also the simultaneous update of all layers. For example, from the chain rule, $\frac{\partial C}{\partial W_{11}^{(2)}} = a_1^{(1)} \frac{\partial a_1^{(2)}}{\partial z_1^{(2)}} \sum_{i=1}^4 W_{i,1}^{(3)} \frac{\partial a_i^{(3)}}{\partial z_i^{(3)}} \frac{\partial C}{\partial a_i^{(3)}}$. The infinitesimal change in $W_{11}^{(2)}$ is computed under the assumption that its constituent terms don't change as $W_{11}^{(2)}$ undergoes this change. However, this assumption doesn't hold given that $\mathbf{a}^{(2)}$, $\mathbf{a}^{(3)}$ and, ultimately, C are themselves dependent on $W_{11}^{(2)}$ and also that $W_{11}^{(2)}$ and all $W_{i,1}^{(3)}$ are updated simultaneously.

■ Shortcut Connections: The Residual Layer

Adding shortcut (skip) connections to layers results in multiple paths of different lengths (including the direct/identity path) between the input and the output.

In general, gradients through shorter paths will be better behaved. Since both the identity term and various short chains of derivatives will contribute to the derivative for each layer, networks with shortcut connections suffer less from shattered gradients.

Batch Normalization

The learning algorithm can also greatly benefit from learning the moments of distribution (i.e., the mean and variance) of the elements of the input to each layer alongside the weights and biases of that layer.

The batch normalization (BN) procedure, in the first step, standardizes each element of the layer's input with the mean and variance that is computed over the current batch for that element and, in the second step, scales and shifts the standardized elements with learned parameters.

In the training phase, for $a_k^{(l-1)}$, the BN procedure computes $\mu_{k,B}$ and $\sigma_{k,B}^2$ from the entire batch and then transforms it to $a_k^{(l-1)} = \gamma_k a_k^{(l-1)} + \beta_k$ in which $a_k^{(l-1)} = \frac{1}{\sigma_{k,B}} (a_k^{(l-1)} - \mu_{k,B})$.

In the inference phase, $a_k^{(l-1)}$ is computed using the average of the values computed for $\mu_{k,B}$ and $\sigma_{k,B}^2$ during the training.

Deep Learning: Frameworks



- Developed by the Google Brain team
- First released in 2015
- TensorFlow 2 was released in 2019



- Developed at the Facebook's AI Research (FAIR) lab
- First released in 2018

Deep Learning: Experiment Setup in TensorFlow 2

Training

```

...
import tensorflow as tf
...

1 seq_network_1=get_seq_network_1(arg_1, ...)

2 model          =ml_algorithm(seq_network_1=seq_network_1,
                               ...)

model.compile(optimizer =tf.keras.optimizers.Adam(...),
              loss      =...,
              <metrics> =...)

3 callbacks=[callback_custom_checkpoint  (),
             callback_custom_monitor     (...),
             callback_custom_history     (...),
             tf.keras.callbacks.Tensorboard(...)]

4 save_data_to_tfrecords_format (arg_1,...)

5 data_train      =read_tfrecords(mode='train'      ,...)
  data_validation=read_tfrecords(mode='validation',...)

model.fit(x          =data_train      ,
          validation_data=data_validation,
          callbacks   =callbacks      ,
          ...)

```

Inference

```

...
import tensorflow as tf
...
model          =tf.keras.models.load_model(path_to_checkpoint,
                                           ...)

data          = ...
predictions=model(data)
...

```

2

```

...
import tensorflow as tf
...

class ml_algorithm(tf.keras.Model):

    def __init__(self,seq_network_1,...):
        super().__init__()
        self.network_1=seq_network_1
        ...

    def compile(self,optimizer,loss,<metrics>):
        super().compile()
        self.optimizer=optimizer
        self.loss      =loss
        self.<metrics>=<metrics>

    def call(self,X,training=None):
        x=self.<network>(X,training=training)
        ...
        return ...

    def train_step(self,data):
        if isinstance(data,tuple):
            x,y=data
            ...
            with tf.GradientTape() as tape:
                predictions=self (x,training=True)
                ...
                loss        =self.loss(y,predictions )
                gradients   =tape.gradient ( loss      , self.<network>.trainable_weights )
                self.optimizer.apply_gradients(zip(gradients, self.<network>.trainable_weights))
            return {'loss': loss,
                    ...
                    }

```

1

```

...
import tensorflow as tf
...

def get_seq_network(arg_1,...):
    X      =tf.keras.Input(shape=..., ...)
    ...
    x      =tf.keras.layers.Dense          (...)(X)
    x      =tf.keras.layers.BatchNormalization(...)(x)
    x      =tf.keras.layers.ReLU           (...)(x)
    ...
    return tf.keras.Model(inputs=X, outputs=x, name=...)

3
...
import tensorflow as tf
...

class callback_custom_checkpoint(tf.keras.callbacks.Callback):

    def on_epoch_end(self, epoch, logs=None):
        tf.keras.models.save_model(self.model ,path_to_checkpoint)

```

Deep Learning: Experiment Setup in TensorFlow 2

④

```
...
import tensorflow as tf
...

def save_data_to_tfrecords_format(...):

    def serialize_example(x,y):
        feature          ={'x': tf.train.Feature(bytes_list=tf.train.BytesList(value=[x.tostring()])),
                            'y': tf.train.Feature(int64_list=tf.train.Int64List(value=[y          ]))}
        example_proto=tf.train.Example(features=tf.train.Features(feature=feature))
        return example_proto.SerializeToString()

    def write_serialized_example(x,y,mode):
        filename =...
        with tf.io.TFRecordWriter(filename) as writer:
            for i in range(number_of_examples):
                example=serialize_example(x[i],y[i])
                writer.write(example)

    def get_train_validation_splits(x,y,...):
        ...
        return x_train, y_train, x_validation, y_validation

    x,y          =get_data          (... )
    x_train, y_train, x_validation, y_validation=get_train_validation_splits(x,y,...)
    write_serialized_example(x_train, y_train, 'train')
    write_serialized_example(x_validation, y_validation, 'validation')
```

⑤

```
...
import tensorflow as tf
...

def read_tfrecords(mode, ...):
    ...
    feature ={'x': tf.io.FixedLenFeature([],tf.string),
              'y': tf.io.FixedLenFeature([],tf.int64 ) }
    def parse_function(example_proto):
        parsed_example=tf.io.parse_single_example(example_proto,feature          )
        x              =tf.io.decode_raw          (parsed_example['x'],tf.float32)
        x.set_shape(shape_of_x)
        y              =parsed_example['y']
        ...
        return x,y
    dataset =tf.data.TFRecordDataset(filenames          )
    dataset =dataset.map          (parse_function          )
    dataset =dataset.batch          (batch_size          , drop_remainder          =True)
    dataset =dataset.shuffle          (buffer_capacity, reshuffle_each_iteration=True)
    return dataset
```

Experiment-I:

Emulation of the FDTD using RNNs

Why Finite Difference Time Domain?

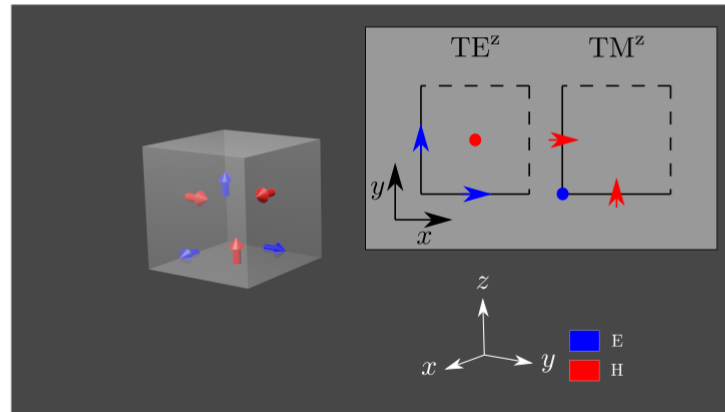
■ The aim of this experiment is to construct a data-driven model for dynamical systems such as a propagating EM wave. In other words, the aim is to construct a model that can produce the next state of a propagating wave given its past states by learning wave dynamics from a dataset of propagating waves.

■ There are two numerical schemes for solving the PDEs that govern the electromagnetic phenomena.

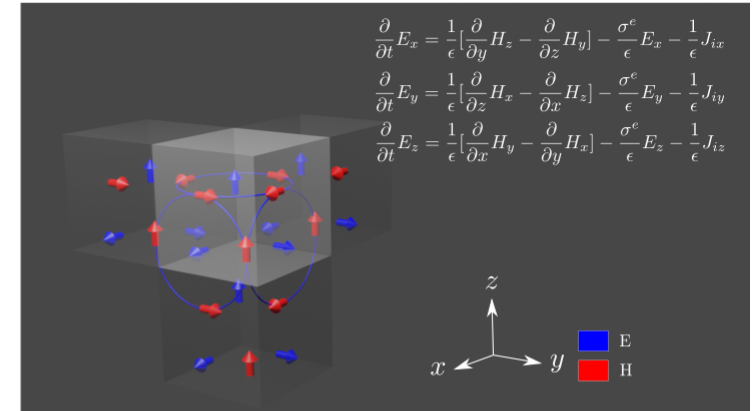
The **Finite Elements Method** (FEM) solves the time-harmonic second-order PDEs for the vector potentials. The FEM discretizes the computational domain into finite elements and interpolates the unknown function over each element using its values at the nodes of that element. It constructs and solves a linear system to find the unknown function's values at the nodes.

The **Finite Difference Time Domain** (FDTD) method solves the time-dependent first-order Maxwell's PDEs. The FDTD method is a time marching scheme that directly forwards the electromagnetic fields in time over a regular spatial grid, referred to as the **Yee grid**.

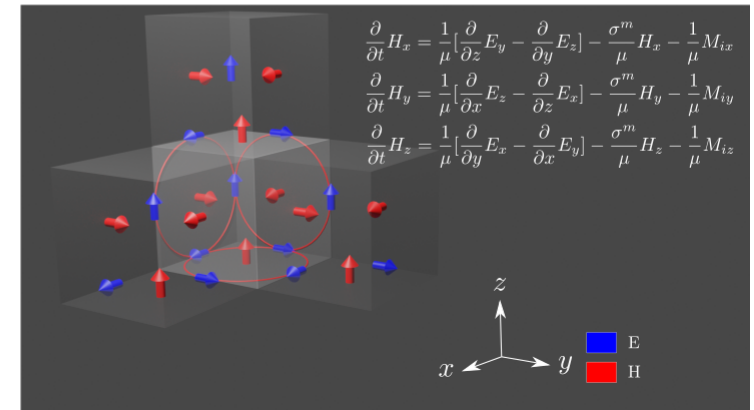
Yee Cell



Yee Grid



$$\begin{aligned}\frac{\partial E_x}{\partial t} &= \frac{1}{\epsilon} \left[\frac{\partial}{\partial y} H_z - \frac{\partial}{\partial z} H_y \right] - \frac{\sigma^e}{\epsilon} E_x - \frac{1}{\epsilon} J_{ix} \\ \frac{\partial E_y}{\partial t} &= \frac{1}{\epsilon} \left[\frac{\partial}{\partial z} H_x - \frac{\partial}{\partial x} H_z \right] - \frac{\sigma^e}{\epsilon} E_y - \frac{1}{\epsilon} J_{iy} \\ \frac{\partial E_z}{\partial t} &= \frac{1}{\epsilon} \left[\frac{\partial}{\partial x} H_y - \frac{\partial}{\partial y} H_x \right] - \frac{\sigma^e}{\epsilon} E_z - \frac{1}{\epsilon} J_{iz}\end{aligned}$$

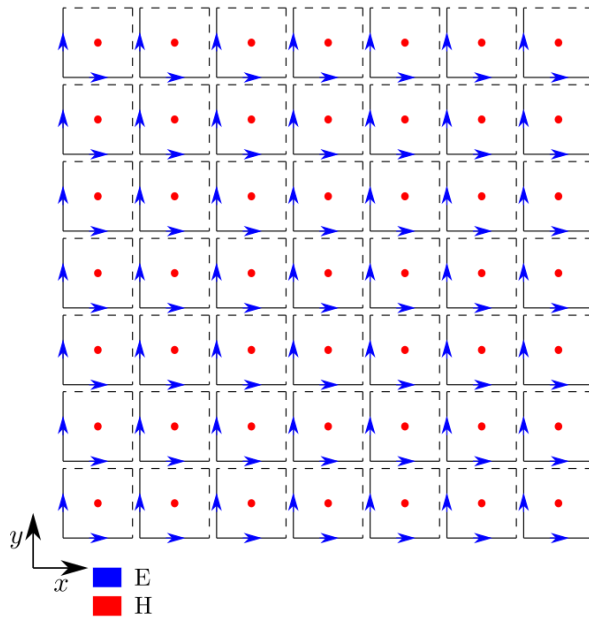


$$\begin{aligned}\frac{\partial H_x}{\partial t} &= \frac{1}{\mu} \left[\frac{\partial}{\partial z} E_y - \frac{\partial}{\partial y} E_z \right] - \frac{\sigma^m}{\mu} H_x - \frac{1}{\mu} M_{ix} \\ \frac{\partial H_y}{\partial t} &= \frac{1}{\mu} \left[\frac{\partial}{\partial x} E_z - \frac{\partial}{\partial z} E_x \right] - \frac{\sigma^m}{\mu} H_y - \frac{1}{\mu} M_{iy} \\ \frac{\partial H_z}{\partial t} &= \frac{1}{\mu} \left[\frac{\partial}{\partial y} E_x - \frac{\partial}{\partial x} E_y \right] - \frac{\sigma^m}{\mu} H_z - \frac{1}{\mu} M_{iz}\end{aligned}$$

Experiment-I: Emulation of the FDTD using RNNs

Why Finite Difference Time Domain?

Update Equations for 2D TE^z Yee Grid



$$H_z^{n+\frac{1}{2}}(i, j) = C_{hh}(i, j)H_z^{n-\frac{1}{2}}(i, j) + C_{hex}(i, j)[E_x^n(i, j+1) - E_x^n(i, j)] + C_{hey}(i, j)[E_y^n(i+1, j) - E_y^n(i, j)] + C_{hm}(i, j)M_z^n(i, j)$$

$$E_x^{n+1}(i, j) = C_{exe}(i, j)E_x^n(i, j) + C_{exh}(i, j)[H_z^{n+\frac{1}{2}}(i, j) - H_z^{n+\frac{1}{2}}(i, j-1)] + C_{exj}(i, j)J_x^{n+\frac{1}{2}}(i, j)$$

$$E_y^{n+1}(i, j) = C_{eye}(i, j)E_y^n(i, j) + C_{eyh}(i, j)[H_z^{n+\frac{1}{2}}(i, j) - H_z^{n+\frac{1}{2}}(i-1, j)] + C_{eyj}(i, j)J_y^{n+\frac{1}{2}}(i, j)$$

$$C_{hh}(i, j) = \frac{2\mu(i, j) - \Delta t\sigma^m(i, j)}{2\mu(i, j) + \Delta t\sigma^m(i, j)}$$

$$C_{hex}(i, j) = \frac{2\Delta t}{[2\mu(i, j) + \Delta t\sigma^m(i, j)]\Delta y}$$

$$C_{hey}(i, j) = -\frac{2\Delta t}{[2\mu(i, j) + \Delta t\sigma^m(i, j)]\Delta x}$$

$$C_{hm}(i, j) = -\frac{2\Delta t}{2\mu(i, j) + \Delta t\sigma^m(i, j)}$$

$$C_{exe}(i, j) = \frac{2\epsilon(i, j) - \Delta t\sigma^e(i, j)}{2\epsilon(i, j) + \Delta t\sigma^e(i, j)}$$

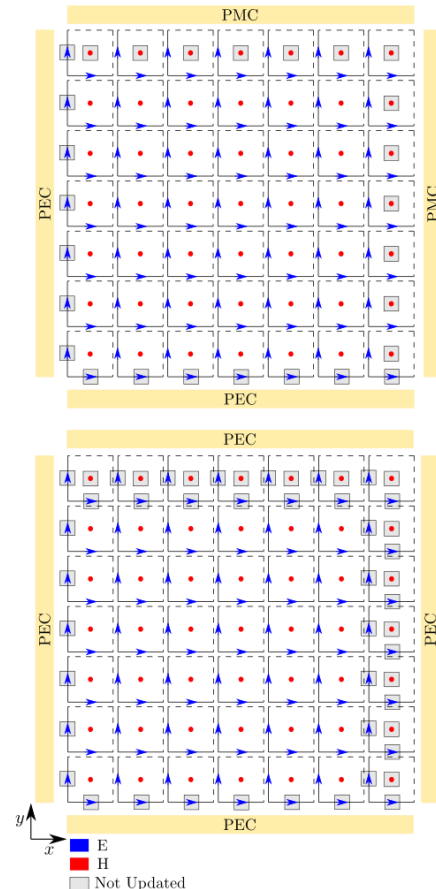
$$C_{exh}(i, j) = \frac{2\Delta t}{[2\epsilon(i, j) + \Delta t\sigma^e(i, j)]\Delta y}$$

$$C_{exj}(i, j) = -\frac{2\Delta t}{2\epsilon(i, j) + \Delta t\sigma^e(i, j)}$$

$$C_{eye}(i, j) = \frac{2\epsilon(i, j) - \Delta t\sigma^e(i, j)}{2\epsilon(i, j) + \Delta t\sigma^e(i, j)}$$

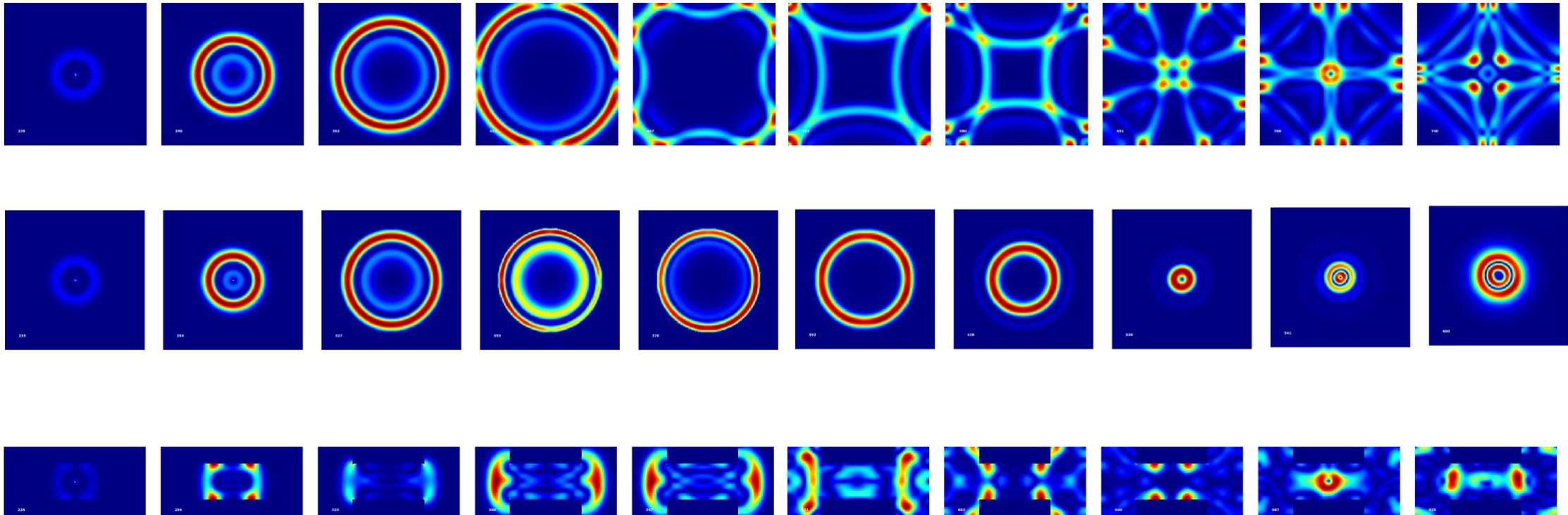
$$C_{eyh}(i, j) = -\frac{2\Delta t}{[2\epsilon(i, j) + \Delta t\sigma^e(i, j)]\Delta x}$$

$$C_{eyj}(i, j) = -\frac{2\Delta t}{2\epsilon(i, j) + \Delta t\sigma^e(i, j)}$$



Experiment-I: Emulation of the FDTD using RNNs Why Finite Difference Time Domain?

2D TE^z : Rectangular, Circular and Ridged Rectangular Cavities



Experiment-I: Emulation of the FDTD using RNNs Why Finite Difference Time Domain?

Media with Anisotropy in Conductivities: 2D TE^z Mode

Maxwell's Equations

$$\begin{aligned} E_x &= -j \frac{1}{\omega \epsilon S_y^e} \frac{\partial H_z}{\partial y} & S_y^e &\equiv 1 - j \frac{\sigma_y^e}{\omega \epsilon} \\ E_y &= j \frac{1}{\omega \epsilon S_x^e} \frac{\partial H_z}{\partial x} & S_x^e &\equiv 1 - j \frac{\sigma_x^e}{\omega \epsilon} \\ H_{zx} &= j \frac{1}{\omega \mu S_x^m} \frac{\partial E_y}{\partial x} & S_x^m &\equiv 1 - j \frac{\sigma_x^m}{\omega \mu} \\ H_{zy} &= -j \frac{1}{\omega \mu S_y^m} \frac{\partial E_x}{\partial y} & S_y^m &\equiv 1 - j \frac{\sigma_y^m}{\omega \mu} \\ H_z &= H_{zx} + H_{zy} \end{aligned}$$

Solutions

$$\begin{aligned} \mathbf{E} &= \frac{E_0}{W} \left(-\sqrt{\frac{S_y^m}{S_y^e}} \sin \theta \mathbf{a}_x + \sqrt{\frac{S_x^m}{S_x^e}} \cos \theta \mathbf{a}_y \right) e^{-j \mathbf{k} \cdot \mathbf{r}} \\ \mathbf{H} &= \left(\frac{1}{Z} E_0 \mathbf{a}_z \right) e^{-j \mathbf{k} \cdot \mathbf{r}} \\ \mathbf{k} &= \omega \sqrt{\mu \epsilon} \left(\sqrt{S_x^e S_x^m} \cos \theta \mathbf{a}_x + \sqrt{S_y^e S_y^m} \sin \theta \mathbf{a}_y \right) \\ Z &= \sqrt{\frac{\mu}{\epsilon}} W \\ W &\equiv \sqrt{\frac{S_x^m}{S_x^e} \cos^2 \theta + \frac{S_y^m}{S_y^e} \sin^2 \theta} \end{aligned}$$

Matched Medium

Matching Condition

$$\begin{aligned} \sigma_x^m &= \frac{\mu}{\epsilon} \sigma_x^e \\ \sigma_y^m &= \frac{\mu}{\epsilon} \sigma_y^e \end{aligned}$$

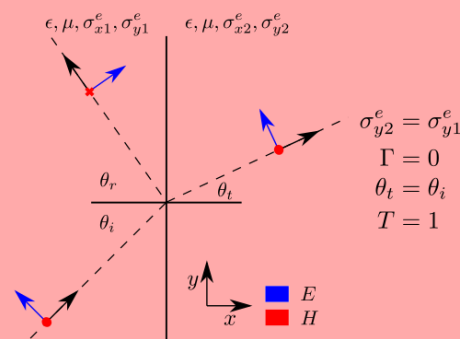
Uniform Plane Wave

$$\begin{aligned} \mathbf{E} &= \mathbf{E}_0 e^{-\sqrt{\frac{\mu}{\epsilon}} [\sigma_x^e \cos(\theta) x + \sigma_y^e \sin(\theta) y]} e^{-j \omega \sqrt{\mu \epsilon} \mathbf{a}_p \cdot \mathbf{r}} \\ \mathbf{H} &= \frac{1}{\eta} \mathbf{a}_p \times \mathbf{E} \end{aligned}$$

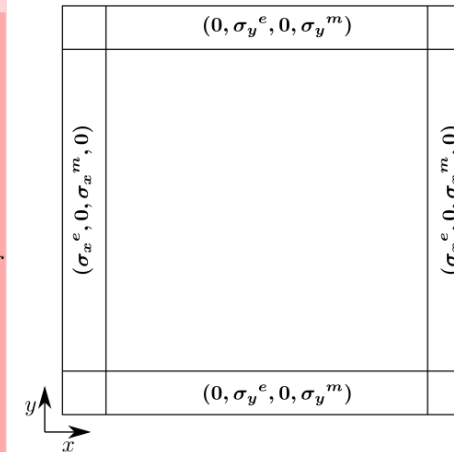
$$\mathbf{E}_0 \equiv E_0 (-\sin \theta \mathbf{a}_x + \cos \theta \mathbf{a}_y)$$

$$\mathbf{a}_p \equiv \cos \theta \mathbf{a}_x + \sin \theta \mathbf{a}_y$$

Reflection & Transmission



Split-PML ABC: 2D TE^z Mode



Electric Conductivity Profile

Left/Bottom

$$\sigma(n) = \sigma_{max} \left(\frac{n_e - n}{n_e - n_b} \right)^P$$

Right/Top

$$\sigma(n) = \sigma_{max} \left(\frac{n - n_b}{n_e - n_b} \right)^P$$

$$\sigma_{max} = -\sqrt{\frac{\epsilon}{\mu}} \frac{(P+1) \ln(R_0)}{2N\Delta}$$

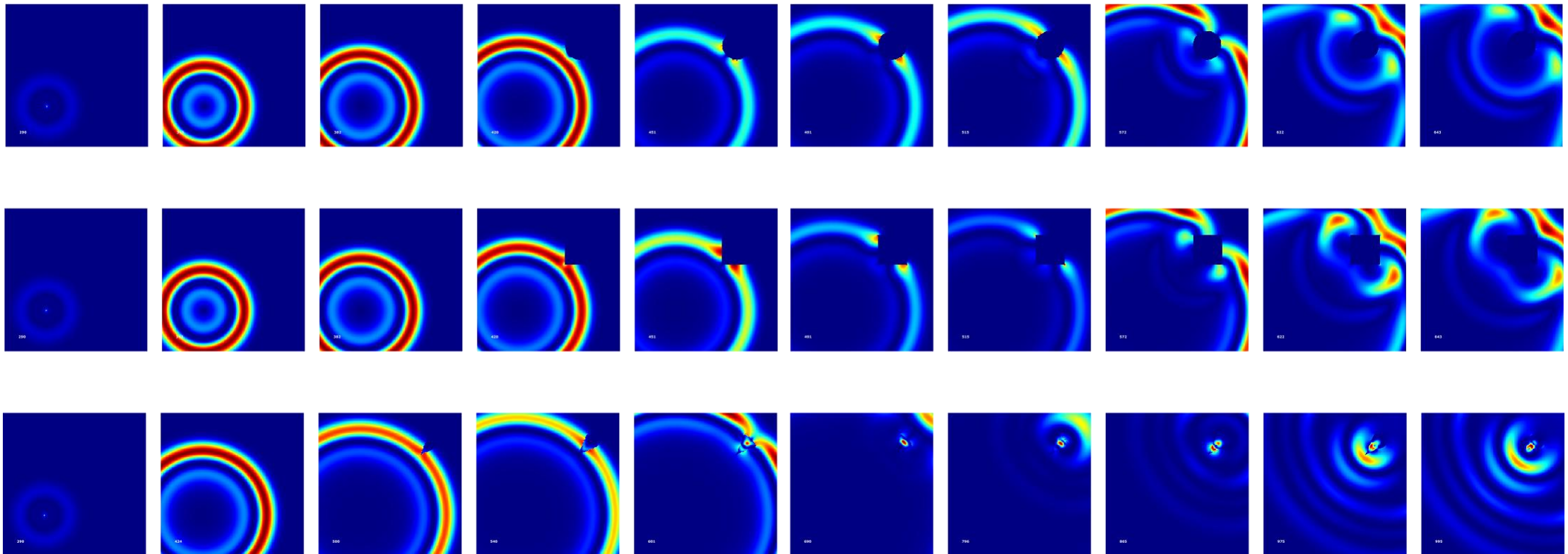
n_b is the beginning index of the PML region
 n_e is the ending index of the PML region
 R_0 is the Required Attenuation
 N is the number of cells in the PML region
 Δ is the cell size

Modified H_z Update

$$\begin{aligned} H_{zx}^{n+\frac{1}{2}}(i, j) &= C_{hh}(i, j) H_{zx}^{n-\frac{1}{2}}(i, j) + C_{hey}(i, j) [E_y^n(i+1, j) - E_y^n(i, j)] \\ C_{hh}(i, j) &= \frac{2\mu(i, j) - \Delta t \sigma_x^m(i, j)}{2\mu(i, j) + \Delta t \sigma_x^m(i, j)} \\ C_{hey}(i, j) &= -\frac{2\Delta t}{[2\mu(i, j) + \Delta t \sigma_x^m(i, j)] \Delta x} \\ H_{zy}^{n+\frac{1}{2}}(i, j) &= C_{hh}(i, j) H_{zy}^{n-\frac{1}{2}}(i, j) + C_{hex}(i, j) [E_x^n(i, j+1) - E_x^n(i, j)] \\ C_{hh}(i, j) &= \frac{2\mu(i, j) - \Delta t \sigma_y^m(i, j)}{2\mu(i, j) + \Delta t \sigma_y^m(i, j)} \\ C_{hex}(i, j) &= \frac{2\Delta t}{[2\mu(i, j) + \Delta t \sigma_y^m(i, j)] \Delta y} \\ H_z^{n+\frac{1}{2}}(i, j) &= H_{zx}^{n+\frac{1}{2}}(i, j) + H_{zy}^{n+\frac{1}{2}}(i, j) \end{aligned}$$

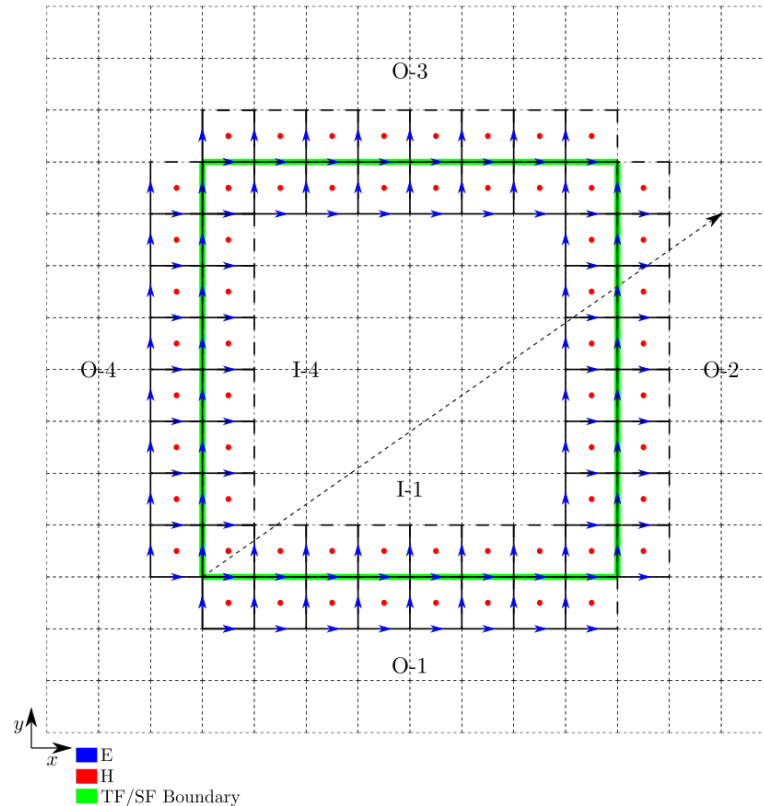
Experiment-I: Emulation of the FDTD using RNNs Why Finite Difference Time Domain?

2D TE^z : Scattering from PEC and Dielectric Objects



Experiment-I: Emulation of the FDTD using RNNs Why Finite Difference Time Domain?

TF/SF Excitation for 2D TE^z Mode



Consistency Updates

O-1 Region

$$H_z^{n+\frac{1}{2}}(i, j) = H_z^{n+\frac{1}{2}}(i, j) - C_{hex}(i, j)E_{x, inc}^n(i, j + 1)$$

O-2 Region

$$E_y^{n+1}(i, j) = E_y^{n+1}(i, j) + C_{eyh}(i, j)H_z^{n+\frac{1}{2}}(i, j)$$

$$H_z^{n+\frac{1}{2}}(i, j) = H_z^{n+\frac{1}{2}}(i, j) + C_{hey}(i, j)E_y^{n+1}(i, j)$$

O-3 Region

$$E_x^{n+1}(i, j) = E_x^{n+1}(i, j) + C_{exh}(i, j)H_z^{n+\frac{1}{2}}(i, j)$$

$$H_z^{n+\frac{1}{2}}(i, j) = H_z^{n+\frac{1}{2}}(i, j) + C_{hex}(i, j)E_{x, inc}^n(i, j)$$

O-4 Region

$$H_z^{n+\frac{1}{2}}(i, j) = H_z^{n+\frac{1}{2}}(i, j) - C_{hey}(i, j)E_y^{n+1}(i + 1, j)$$

I-1 Region

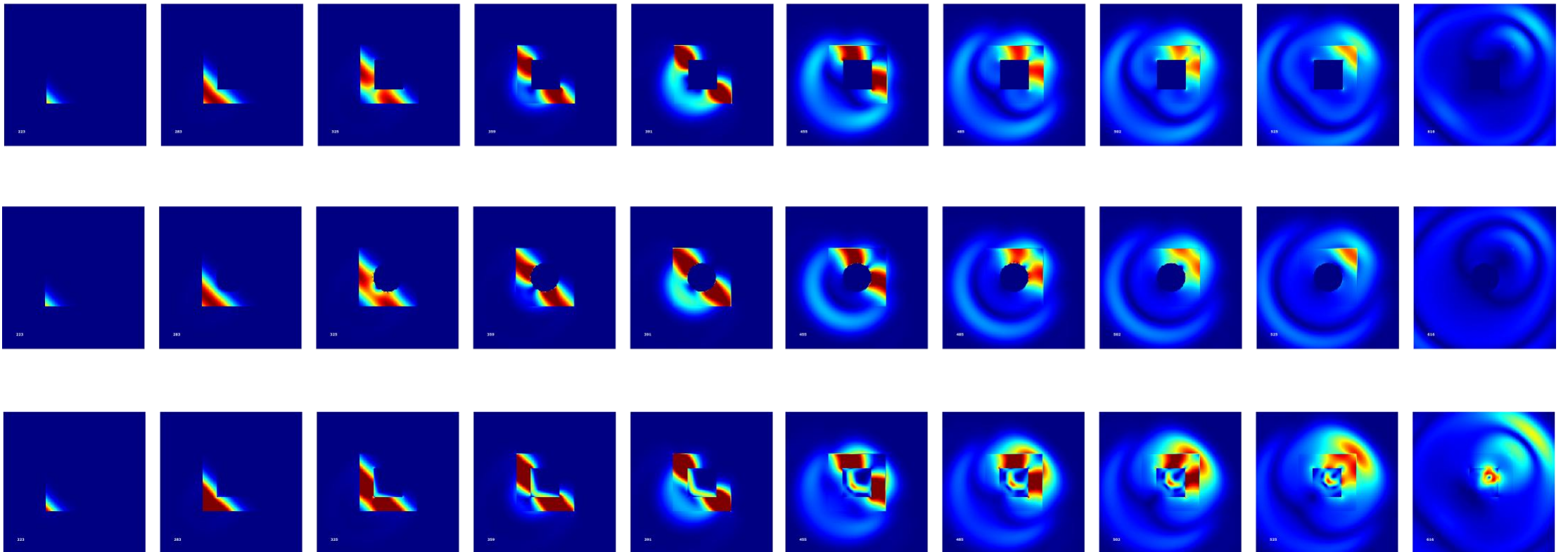
$$E_x^{n+1}(i, j) = E_x^{n+1}(i, j) - C_{exh}(i, j)H_z^{n+\frac{1}{2}}(i, j - 1)$$

I-4 Region

$$E_y^{n+1}(i, j) = E_y^{n+1}(i, j) - C_{eyh}(i, j)H_z^{n+\frac{1}{2}}(i - 1, j)$$

Experiment-I: Emulation of the FDTD using RNNs Why Finite Difference Time Domain?

2D TE^z : Scattering from PEC and Dielectric Objects with TF/SF Excitation



Experiment-I: Emulation of the FDTD using RNNs Why Finite Difference Time Domain?

Recasting FDTD as Video Prediction

■ The main reason that the FDTD method is the suitable numerical scheme for experimentation is that its output is a **sequence of grid values**. The grid of field values can be interpreted as a grid of pixels, i.e., an **image**. Then, the temporal variations of the grid values constitute a sequence of images, i.e., a **video**. This interpretation connects the domain of Computational EM (CEM) with the domain of Computer Vision (CV) and recasts the FDTD scheme for time-marching of EM fields as the task of **predicting the future frames of a video**.

■ With the aim of constructing a **data-driven model of EM dynamics**, we investigate whether FDTD's set of update equations can be substituted with an RNN. In other words, whether an RNN can produce a sequence of grid values, i.e., a video, that evolve as electromagnetic waves. As examples, we consider the problem of plane wave propagation and plane wave scattering from perfect electric conductors (PECs) and set up experiments in which an RNN is first trained with a set of FDTD videos and then, in the inference phase, is presented with an FDTD video (which it has not seen during training) and is asked to produce its next few frames.

Experiment-I:

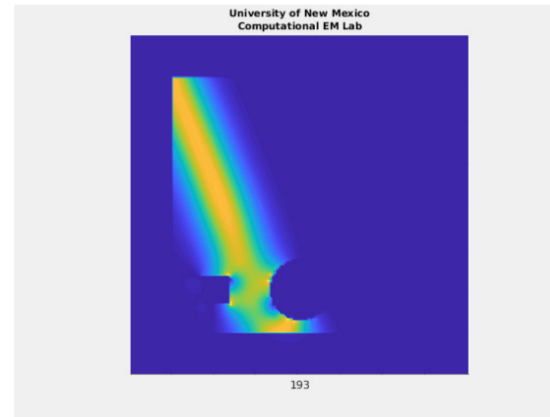
Emulation of the FDTD using RNNs

Dataset: Type-1

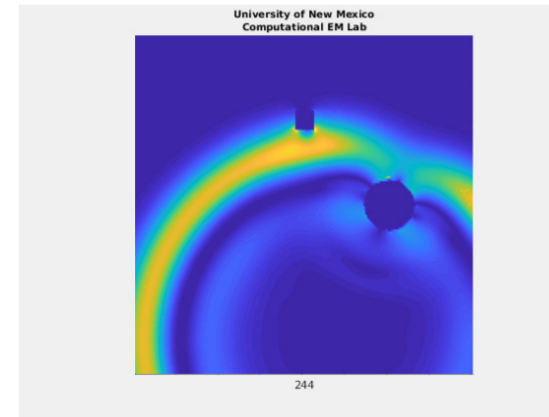
Dataset: Type-2

Dataset: Type-3

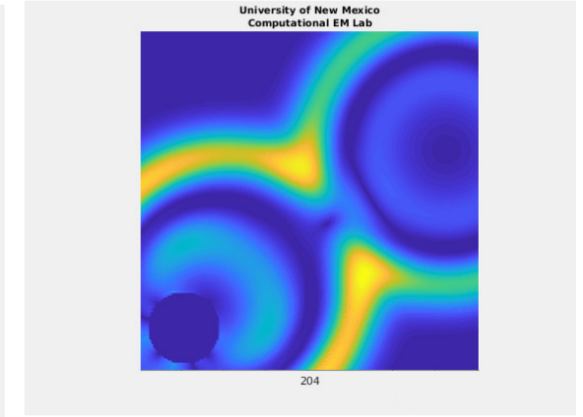
- Each dataset comprises 100 simulations with a Gaussian pulse which has the highest frequency component of 2 GHz. In each simulation all three field components E_x , E_y , H_z are recorded within a region of 128×128 cells for 400 time steps and finally, all the frames in which the excitation hasn't yet fully entered the computational domain are cut from its beginning.
- In each dataset type, 75 simulations are allocated to training and 25 to testing. The size of the training sets depends on the size of the videos (which is a hyperparameter). The total number of simulated frames available for training for dataset Type-1 is 18,750 and for datasets Type-2 and Type-3 is 15,750.



For dataset Type-1, the TF/SF excitation is used to sweep a planar wavefront (angle of propagation chosen randomly between 20° and 70°) across the domain of interest and scattered waves from PEC objects are observed. PEC objects are a random mix of circular and square shapes with sizes chosen randomly between $0.4\lambda_{min}$ to $0.6\lambda_{min}$.



For dataset Type-2, a point source is applied at a random location resulting in a spherical wavefront which then propagates across the domain of interest and scatters from PEC objects. PEC objects are a random mix of circular and square shapes with sizes chosen randomly between $0.4\lambda_{min}$ to $0.6\lambda_{min}$.



For dataset Type-3, two point sources are applied at random locations resulting in a complex wavefront pattern (formed by the sum of the fields from each source at each point in time) which then propagates across the domain of interest and scatters from a circular PEC object of fixed size and fixed location (at the bottom left corner of the domain).

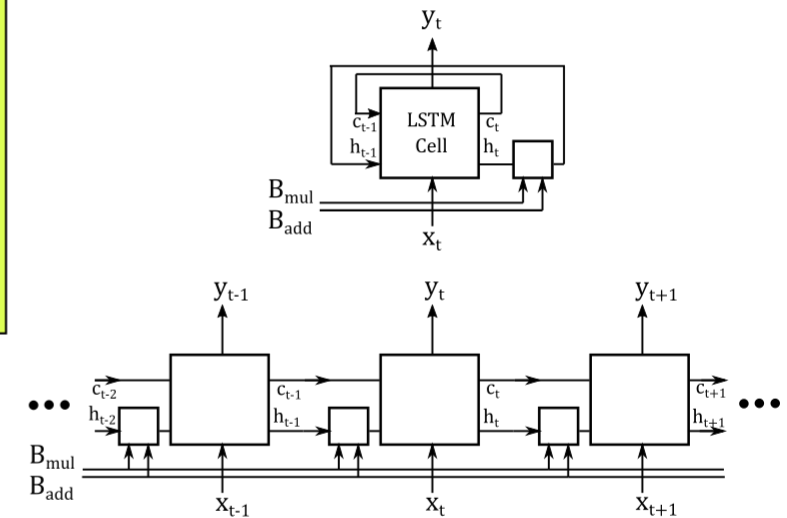
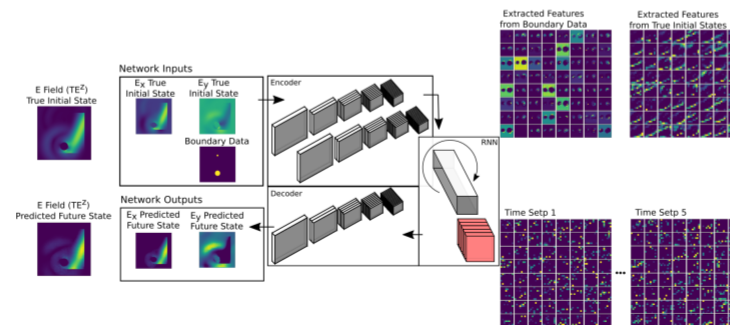
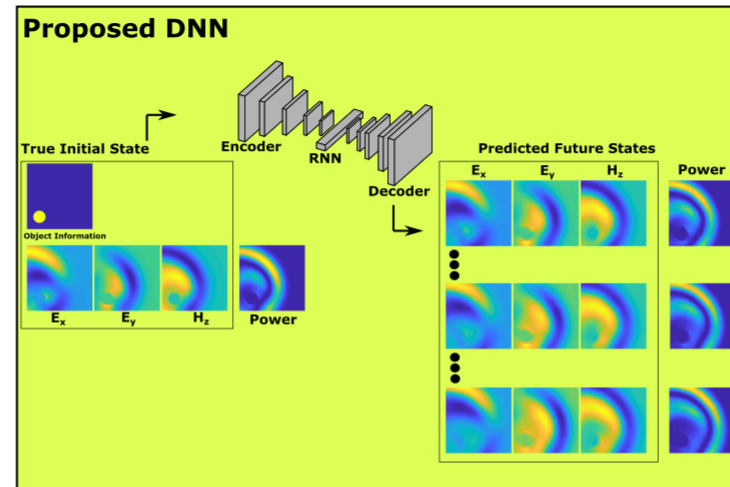
Experiment-I: Emulation of the FDTD using RNNs

Encoder-Recurrent-Decoder (ERD)

- The ERD model comprises a convolutional **encoder**, a convolutional **LSTM** and a convolutional **decoder**.

When a video is presented to the model, the encoder extracts its first frame (or alternatively, its last frame if operating in the Continuous Prediction mode in the inference phase) and reduces that frame to a set of latent space features that are then fed to the LSTM. With the extracted features as initial input, the hidden state of the LSTM is updated for a specific number of times to produce a stack of features that can be interpreted as future frames in the latent space. Finally, the stack of features is fed to the decoder to construct complete future frames of the EM fields for that video.

- To incorporate the object information into the learned representations, and to ensure that the objects remain in their positions while the dynamics of wave propagation and scattering unfold, the model is provided with information about the object in the form of a binary background (1's for locations where the PEC object exists and 0's elsewhere) alongside the input time-domain field data. Furthermore, the LSTM is modified accordingly to incorporate the object information.



Experiment-I: Emulation of the FDTD using RNNs

■ Inference

- **Continuous Prediction** refers to a mode of prediction in which the model predicts multiple frames into the future from a single initial true frame by continuously feeding its output back to its input.
- **Test Prediction** refers to a mode of prediction in which the model predicts only the next frame from the true frame, i.e., for each prediction it requires the previous true frame.

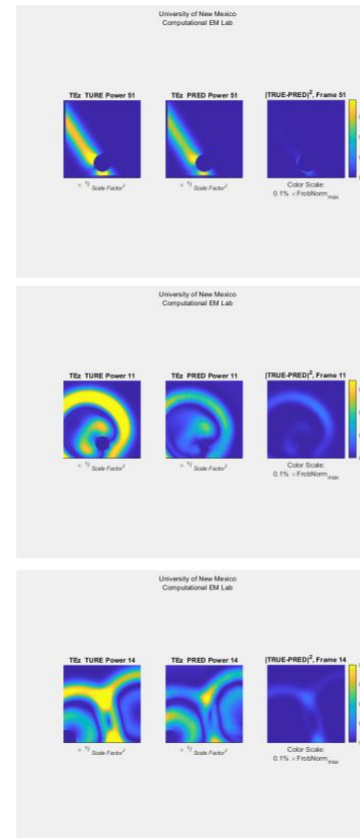
■ Error

For each frame (i.e., time-step), denoted by f_t , the error is defined as $|f_t^{\text{FDTD}} - f_t^{\text{ERD}}|^2$. The color scale on error plots is clamped to a fraction of the quantity denoted by $N_{\text{Frob,max}}^{\text{ERD}}$ and computed as $N_{\text{Frob,max}}^{\text{ERD}} = \max(\|f_t^{\text{FDTD}} - f_t^{\text{ERD}}\|_{\text{Frob}})$.

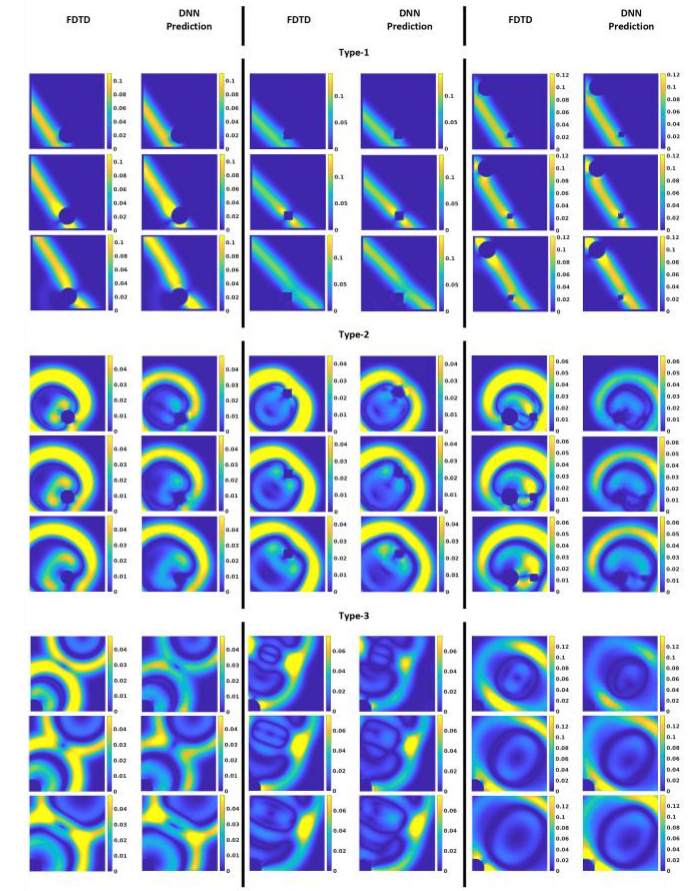
■ Speedup

Inference latency is 22 ms. A single update of the in-house developed FDTD solver takes 373 ms. The ERD model exhibits a $17\times$ speedup.

Continuous Prediction



Test Prediction



Experiment-I: Emulation of the FDTD using RNNs

DDM-based Parallelization

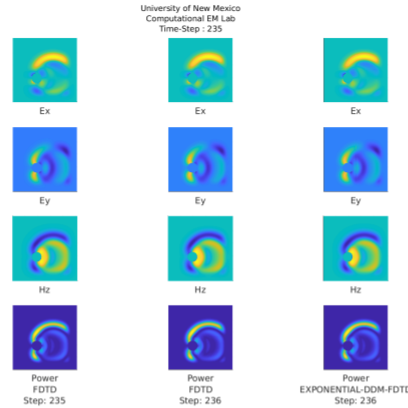
■ We propose to use the ERD model as a computation tool in a **Domain Decomposition Method** (DDM) based parallelization scheme. In this scheme, the entire computational domain is divided into smaller regions, called **sub-domains**, and computations are carried out in parallel over each. Since events that occur in each sub-domain are similar to the ones that occur in others, e.g., a wavefront propagates from one corner to another or a wavefront impinges on an object and scatters, a data-driven model of EM dynamics can be used for generating the future states of all sub-domains.

■ To implement this scheme (1) cast Maxwell's equations in the form of a **system of linear equations**. The solution can be expressed as the multiplication of the **matrix exponential** with the vector of **initial conditions**. (2) choose a **block-diagonal** matrix as the **pre-conditioner**. With this choice, the matrix-vector multiplication involving a large matrix exponential is transformed into a set of parallel matrix-vector multiplications involving smaller matrix exponentials. (3) use a **generalized Runge Kutta** (RK) method for integration.

■ Speedup

The run time of the DDM parallelization scheme exhibits $2\times$ speedup when the ERD model is used for subdomain computations compared to its pure form which uses matrix exponential operations.

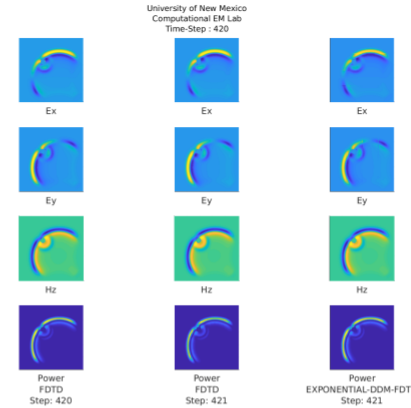
Dataset: Type-4



This Dataset contains two-time-step simulations of size 256×256 (which can be divided into 4 subdomains of size 128×128). A single point source excites a spherical wavefront which scatters from a single circular PEC object of size $0.5\lambda_{min}$ and eventually dissipates in the PML walls. The relative locations of the point source and the PEC object were changed based on all unique combinations for which source and object remained completely in a subdomain.

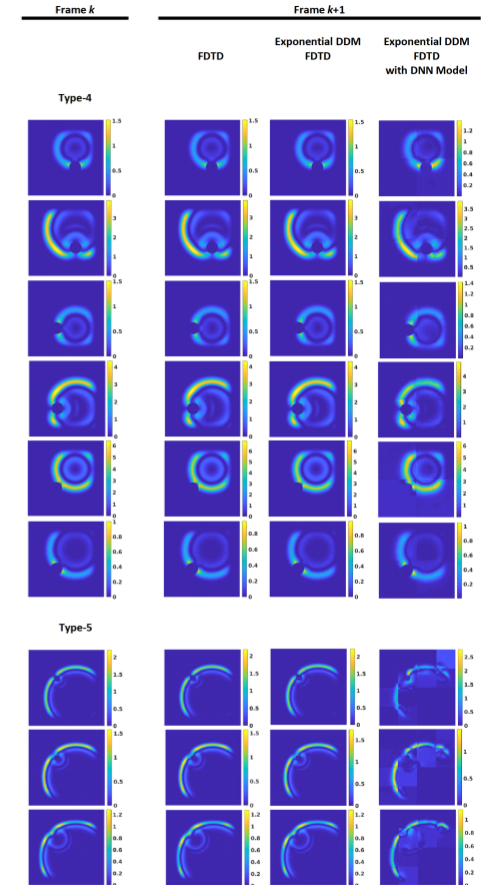
In the interval between frames 180 to 250, every five frames, two frames were recorded resulting in a total of 90 simulations. 67 simulations were randomly selected for training and the rest were assigned to testing.

Dataset: Type-5



This Dataset contains two-time-step simulations of size 512×512 (which can be divided into 16 subdomains of size 128×128). A single point source, the location of which is fixed at the lower right of the domain, excites a spherical wavefront which scatters from a single circular PEC object of size $0.5\lambda_{min}$ (the location of which is fixed within a subdomain near the middle of the domain) and eventually dissipates in the PML walls.

In the interval between frames 320 to 450, every ten frames, two frames were recorded resulting in a total of 14 simulations for testing.



repo: gitlab.com/oameed/unm_cem_dlfdd

Experiment-II:

Emulation of the FDTD using Transformers

Recasting FDTD as Video Prediction

■ The aim of this experiment is to construct a data-driven model for dynamical systems such as a propagating EM wave. In other words, the aim is to construct a model that can produce the next state of a propagating wave given its past states by learning wave dynamics from a dataset of propagating waves.

■ The ERD architecture proved a capable model for emulating the dynamics of EM propagation and scattering. The speedup in computations enabled by its incorporation into the DDM-based parallelization scheme is a demonstrated example of the benefits that ML can bring to conventional computational workflows.

In the context of the DDM-based parallelization as an application, the task of the ERD model is to provide the next time-step in each sub-domain. As a result, it can be argued that any data-driven model that can do next-frame prediction can be utilized for this application. Therefore, in answering the second fundamental question "What are the domain-specific machine learning algorithms required to convert various datasets to modeling knowledge?", the ERD architecture is, certainly, one of the domain-specific ML algorithms that can be used to convert a dataset of propagating and scattering waves into modeling knowledge. However, it might not be the only one.

■ We search for other data-driven models that can function in the same way as (and therefore replace in a modular fashion) the ERD model in the DDM-based parallelization scheme. New architectures have recently been introduced that are also capable of processing sequential data.

The **Transformer** is one such architecture which was developed in the field of NLP for translation tasks. In that context, it takes a segment of text (i.e., a sequence of tokens) in one language and outputs its translation in another (which is itself a sequence of tokens).

■ With the aim of constructing a **data-driven model of EM dynamics**, we investigate whether FDTD's set of update equations can be substituted with a **Transformer**. In other words, whether a Transformer can produce a sequence of grid values, i.e., a video, that evolve as electromagnetic waves.

As examples, we consider the problem of plane wave propagation and plane wave scattering from perfect electric conductors (PECs) and set up experiments in which a Transformer is first trained with a set of FDTD videos and then, in the inference phase, is presented with an FDTD video (which it has not seen during training) and is asked to produce its next few frames.

Experiment-II: Emulation of the FDTD using Transformers

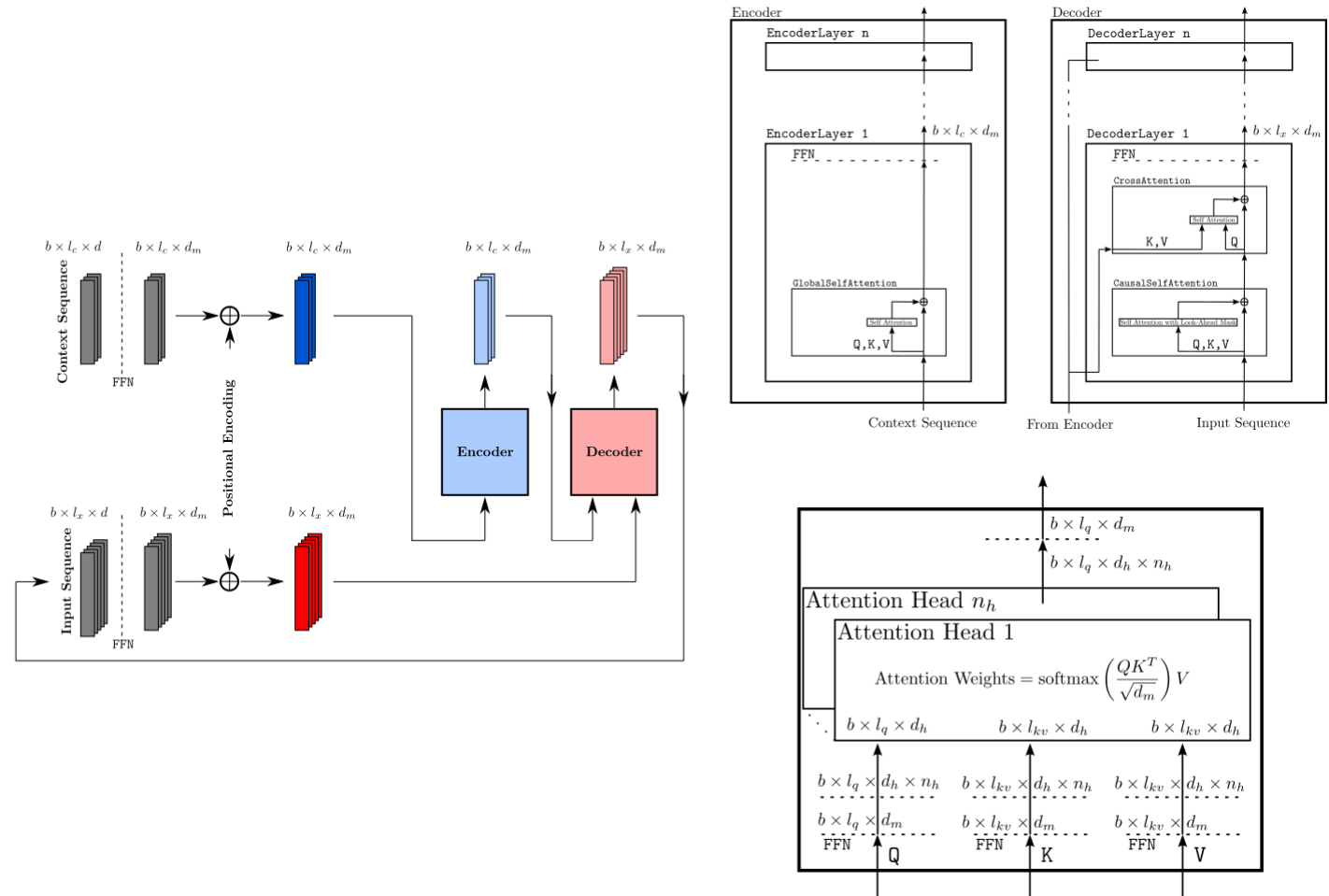
Transformer

- Starting from a sequence of **embeddings**, **positional encoding** is added to give the model information regarding the relative or absolute position of the elements in the sequence. The result is then fed to the **encoder** which will map it to a representation sequence.

Output generation is an auto-regressive process. It starts with feeding the **decoder** with a one-element input sequence along with the encoder's output. The last element in the decoder's output is extracted, embedded, and appended to the input sequence. This process continues with the decoder taking in the entire previously predicted output sequence along with encoder's output in each step.

In each step, the decoder's output sequence has the same number of elements as its input; In the training phase decoder has learned to shift input sequence elements forward by one position. Therefore, in each step of output generation, only the last element in the output sequence contains new information.

- We divide each sequence into **past** and **future** segments. The past segment plays the role of the **context sequence** and is fed to the encoder. Also, its last frame is fed to the decoder as the initial frame for the prediction of the future sequence. With this proposed scheme, we **translate the past into the future** using the Transformer architecture.
- Self-Attention** produces a representation of the relatedness of different positions of a sequence without regard to their distance.



Experiment-II: Emulation of the FDTD using Transformers

Implementation

The hyperparameter values in our implementation are as follows: both the encoder and the decoder consist of 3 layers; $d_m = 256$ and $n_h = 8$.

This configuration results in the largest possible model that we can implement using a single GPU. Larger values for the above mentioned hyperparameters will cause the GPU to throw an OOM (Out Of Memory) error and abort the training. The size of the implemented model with the configuration mentioned above is about 60M (million) parameters.

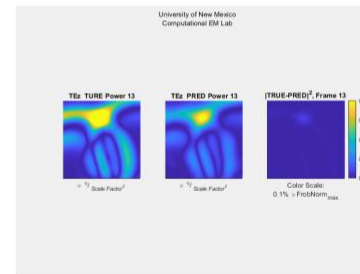
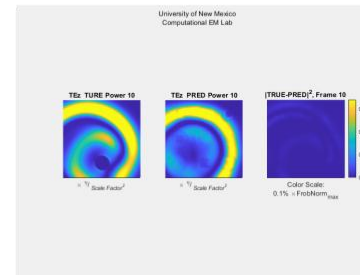
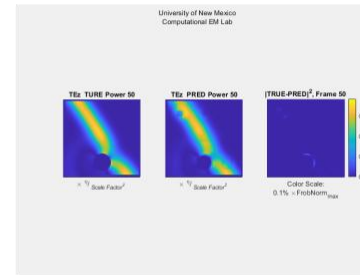
Error

For each frame (i.e., time-step), denoted by f_t , the error is defined as $|f_t^{\text{FDTD}} - f_t^{\text{ERD}}|^2$. The color scale on error plots is clamped to a fraction of the quantity denoted by $N_{\text{Frob,max}}^{\text{ERD}}$ and computed as $N_{\text{Frob,max}}^{\text{ERD}} = \max(\|f_t^{\text{FDTD}} - f_t^{\text{ERD}}\|_{\text{Frob}})$.

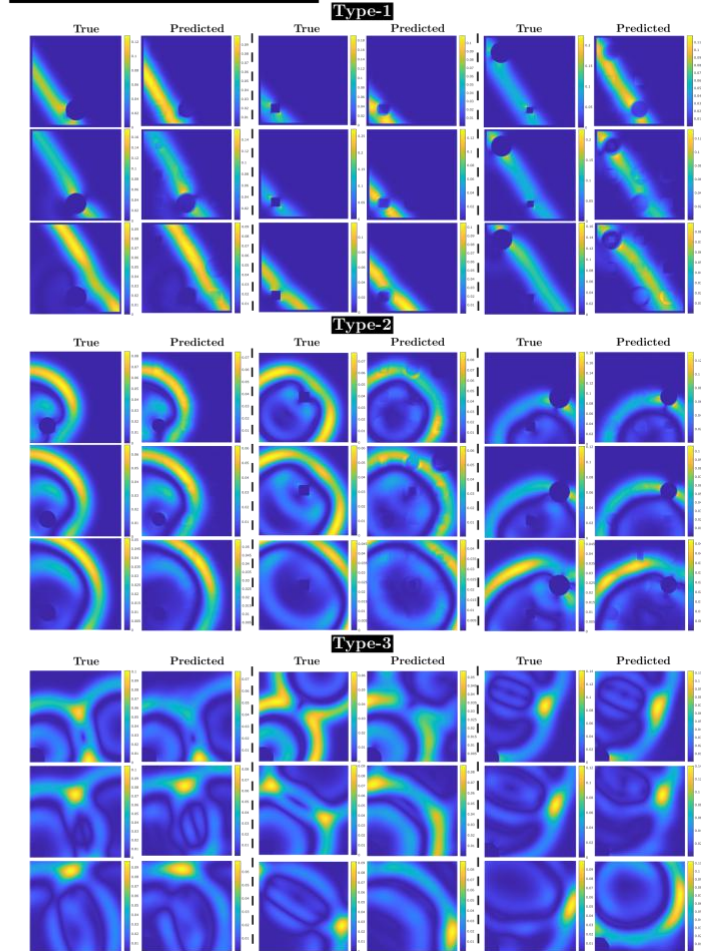
Speedup

For a batch size of 25, the average per-frame inference latency is 25.30 ms. The values of this quantity for the in-house developed FDTD solver and MEEP are 373 ms and 42.61 ms respectively. Therefore, the Transformer exhibits speedups of $14\times$ and $1.5\times$ respectively.

Continuous Prediction



Continuous Prediction



repo: github.com/oameed/unm_cem_trnsf_gnn_fdt

Experiment-III:

Emulation of the FDTD using GNNs

Recasting FDTD as Video Prediction

■ The aim of this experiment is to construct a data-driven model for dynamical systems such as a propagating EM wave. In other words, the aim is to construct a model that can produce the next state of a propagating wave given its past states by learning wave dynamics from a dataset of propagating waves.

■ The ERD architecture proved a capable model for emulating the dynamics of EM propagation and scattering. The speedup in computations enabled by its incorporation into the DDM-based parallelization scheme is a demonstrated example of the benefits that ML can bring to conventional computational workflows.

In the context of the DDM-based parallelization as an application, the task of the ERD model is to provide the next time-step in each sub-domain. As a result, it can be argued that any data-driven model that can do next-frame prediction can be utilized for this application. Therefore, in answering the second fundamental question "What are the domain-specific machine learning algorithms required to convert various datasets to modeling knowledge?", the ERD architecture is, certainly, one of the domain-specific ML algorithms that can be used to convert a dataset of propagating and scattering waves into modeling knowledge. However, it might not be the only one.

■ We search for other data-driven models that can function in the same way as (and therefore replace in a modular fashion) the ERD model in the DDM-based parallelization scheme. New architectures have recently been introduced that are also capable of processing sequential data.

The **Graph Convolutional Network** (GCN) is one such architecture. It constructs representations by updating the attribute vector of each node in the graph using information from its neighbors. Successive operation of a GCN on a graph produces a sequence of attribute vectors.

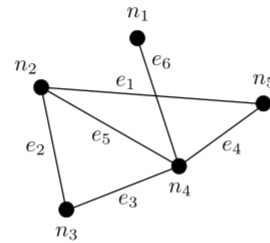
■ With the aim of constructing a **data-driven model of EM dynamics**, we investigate whether FDTD's set of update equations can be substituted with a **GCN**. In other words, whether a GCN can produce a sequence of grid values, i.e., a video, that evolve as electromagnetic waves.

As examples, we consider the problem of plane wave propagation and plane wave scattering from perfect electric conductors (PECs) and set up experiments in which a GCN is first trained with a set of FDTD videos and then, in the inference phase, is presented with an FDTD video (which it has not seen during training) and is asked to produce its next few frames.

Experiment-III: Emulation of the FDTD using GNNs

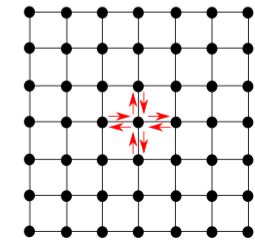
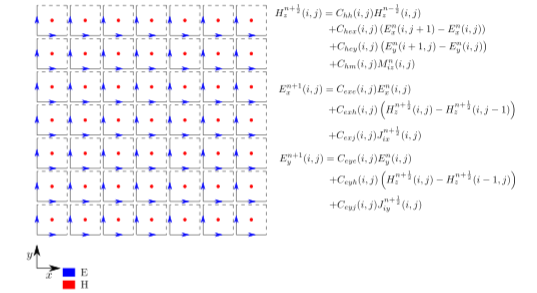
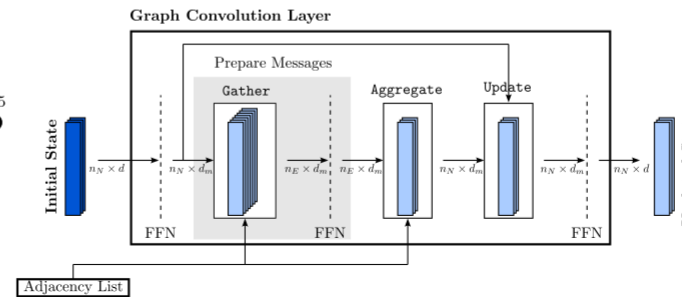
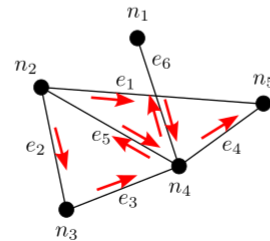
Graph Neural Network

- GNNs were developed to handle graph-structured data such as connectivity in social networks. Unstructured data types such as images can be interpreted as graphs. The pixels can be represented by the **nodes** and the spatial adjacency of the pixels can be represented by **edges** while the red, green, and blue pixel values will constitute the vector of **attributes** for each node.
- The operations of a GCN are as follows: First, each node prepares a **message** for each of its neighbors from its current state of attribute vector and then it passes those messages to their intended recipients. Once messages from all their neighbors are received, each node will **aggregate** them and then use the resulting representation to **update** the current state of its attribute vector.
- Field components in each cell of the Yee grid are updated using information from neighboring cells. We represent the Yee grid as a graph in which nodes are the cells, node attribute vectors comprise the field components and edges connect neighboring cells. As nodes exchange information (in the form of messages) with each application of the GCN to the graph, fields will progress forward in time by one step.



Adjacency List

$e_1 : (n_2, n_5)$
 $e_2 : (n_2, n_3)$
 $e_3 : (n_3, n_4)$
 $e_4 : (n_4, n_5)$
 $e_5^1 : (n_2, n_4)$
 $e_5^2 : (n_4, n_2)$
 $e_6^1 : (n_1, n_4)$
 $e_6^2 : (n_4, n_1)$



Experiment-III: Emulation of the FDTD using GNNs

Implementation

The hyperparameter values in our implementation are as follows: since our frame size is 128×128 pixels, $n_N = 16,384$ and $n_E = 65,024$.

Therefore, we set $d_m = 32$ which is the largest possible model size that we can implement using a single GPU. Larger values for d_m will cause the GPU to throw an OOM (Out Of Memory) error and abort the training. The size of the implemented model with the configuration mentioned above is about 13K parameters.

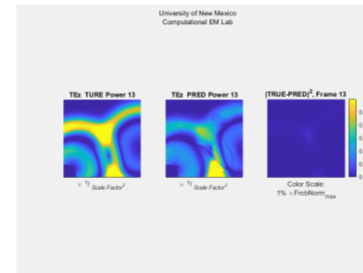
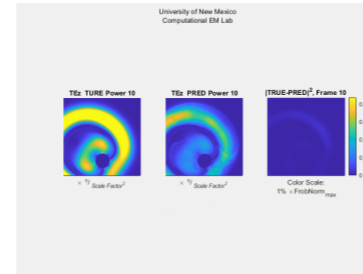
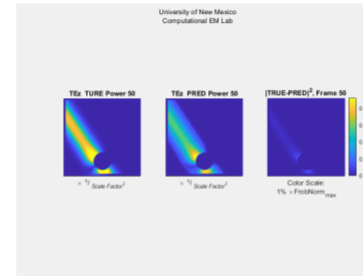
Error

For each frame (i.e., time-step), denoted by f_t , the error is defined as $|f_t^{\text{FDTD}} - f_t^{\text{ERD}}|^2$. The color scale on error plots is clamped to a fraction of the quantity denoted by $N_{\text{Frob,max}}^{\text{ERD}}$ and computed as $N_{\text{Frob,max}}^{\text{ERD}} = \max(\|f_t^{\text{FDTD}} - f_t^{\text{ERD}}\|_{\text{Frob}})$.

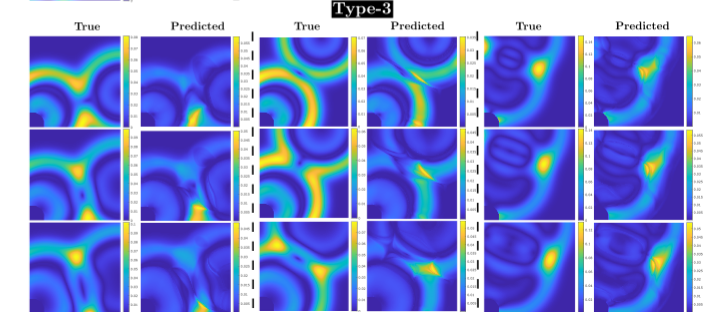
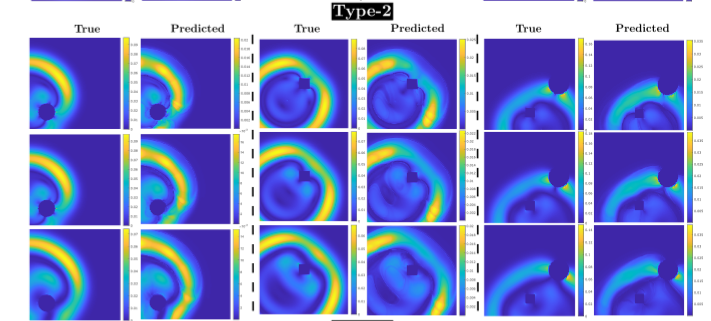
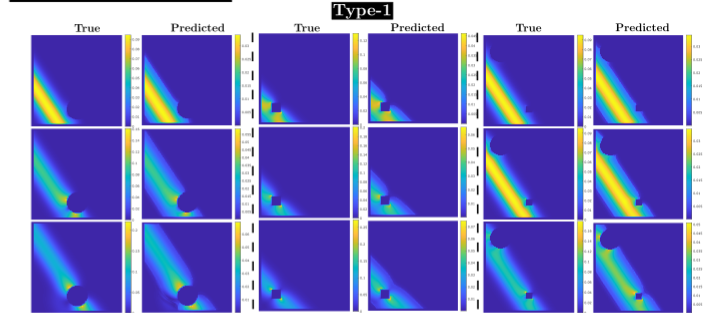
Speedup

The average per-frame inference latency is 43.17 ms. The values of this quantity for the in-house developed FDTD solver and MEEP are 373 ms and 42.61 ms respectively. Therefore, the GNN exhibits a speedup of $9\times$ compared to the former and no speedup compared to the latter.

Test Prediction



Test Prediction



Experiment-IV:

Device Modeling using GANs

Recasting Antenna Design as Synthetic Data Generation

■ The aim of this experiment is to construct a data-driven knowledge database, e.g., one that contains application-specific near-optimal antenna designs. In other words, the aim is to construct a model that can produce new desirable antenna designs by learning the data-generating distribution of a dataset of desirable designs.

■ With the aim of constructing a **data-driven knowledge database**, we investigate whether a Generative Adversarial Network (GAN) can learn the distribution of a dataset of near-optimal antenna designs. In other words, whether a GAN can produce new antenna designs that exhibit a desired response.

As an example, we consider the problem of designing the Log-Periodic Folded Dipole Array (LPFDA) antenna for two non-overlapping ranges of Q-factor values. We represent each antenna with the vector of its structural parameters and consider each desirable range of the Q-factor as a class therefore recasting our problem as the task of **generating new samples from a given class**.

We set up experiments in which a Conditional GAN (CGAN) and a library of vanilla GANs are first trained with a dataset of near-optimal structures and then, in inference phase, are asked to generate new LPFDA designs from each Q-factor range.

Experiment-IV: Device Modeling using GANs

■ The array is designed to operate from 350 MHz to 1 GHz and its elements are printed on a substrate with $\epsilon_r = 2.33$ and $h = 0.786$ mm. The structure is fed from the bottom using a coaxial cable that drives the center conductor. Ten structural parameters were used to form the antenna's vector representation $X = [x_2, x_3, x_4, x_5, c_2, c_3, c_4, c_5, a_1, a_2]^T$ while $x_1, x_6, x_7, c_1, c_6, c_7$ and c_8 were kept constant in the design. Since LPFDA is a wide-band antenna, we define its Q-factor as the average of Q-factors computed at frequencies that are sampled from its operational bandwidth.

■ Datasets rfant-1 and rfant-2 are used to train each vanilla GAN separately while *rfant-3* was used to train the CGAN.

Dataset: rfant-1

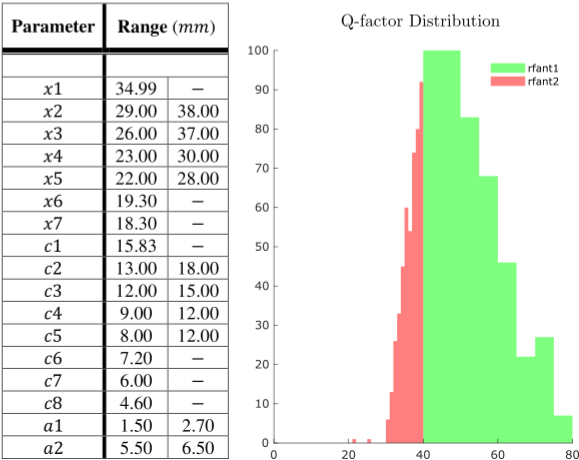
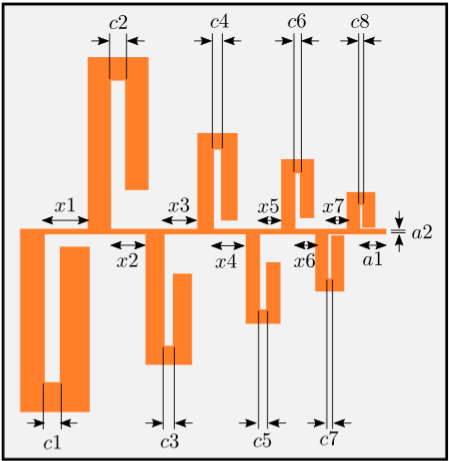
This dataset contains 500 designs for each of which $40 < Q < 80$.

Dataset: rfant-2

This dataset contains 500 designs for each of which $Q < 40$.

Dataset: rfant-3

This dataset is created by the union of rfant-1 and rfant-2, with label 1 assigned to examples from rfant-1 and label 0 assigned to examples from rfant-2.



Experiment-IV:

Device Modeling using GANs

- GANs comprise two models, a **Generator** and a **Discriminator**. The Generator maps a vector of random noise to a vector of features. The Discriminator decides if its input is from the dataset (real) or is produced by the Generator (fake).

In the training phase, the Generator learns to deceive the Discriminator by producing features that increasingly resemble the ones from the dataset. In other words, in the training phase, the Generator learns an estimate of the distribution of the dataset and can map random noise to examples from that dataset. In the inference phase, the Generator will produce new examples that are similar to the ones from the dataset as they are drawn from the same distribution.

- The Vanilla GAN has no mechanism for guiding the generation of datapoints. In other words, in the case of a dataset that comprises multiple categories of examples, the Vanilla GAN can't be instructed to generate datapoints from a specific category.

The Conditional GAN was introduced to address this problem. The Generator of the CGAN accepts a noise vector and a class label as inputs. The label is provided as an integer and is then embedded (i.e., projected onto a higher-dimensional space) and concatenated with the noise vector. This mixture of noise and class information is mapped to a datapoint after passing through the Generator.

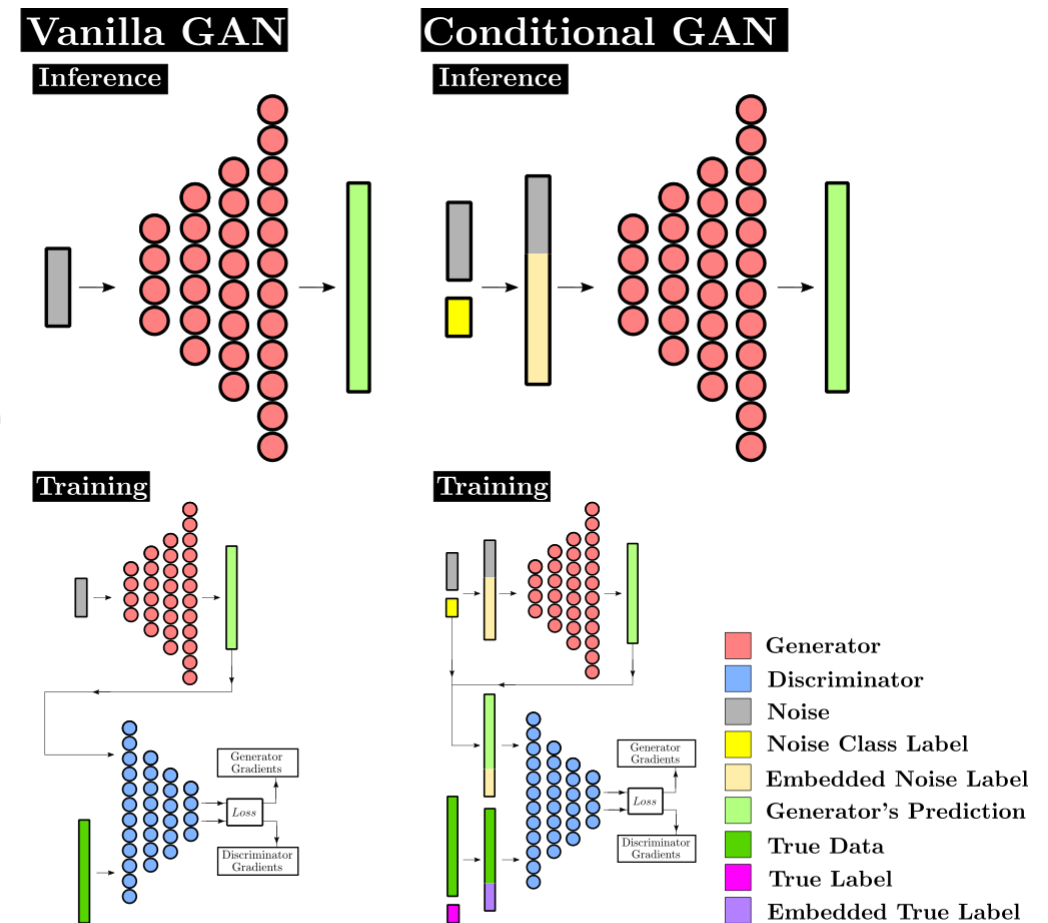
Wasserstein GAN (WGAN)

Training of GANs is significantly improved by (1) replacing the original GAN's loss with Wasserstein loss as a measure of the closeness of the predicted distribution and data distribution (2) constraining the Discriminator to the set of 1-Lipschitz Continuous functions by confining (clipping) its weights to a compact space $[-c, c]$ (3) updating the Discriminator weights to optimality before each update of the generator weights and (4) using RMSprop algorithm for optimization.

WGAN with Gradient Penalty (WGAN-GP)

WGAN's training was further improved by enforcing the 1-Lipschitz Continuity constraint through a Gradient Penalty (GP) instead of weight clipping. In this improved scheme, i.e., WGAN-GP, weight clipping and batch normalization are removed from the Discriminator and a regularizer, i.e., the gradient penalty, is added to the Discriminator's loss and ADAM algorithm is used for optimization.

- We have adopted the WGAN scheme for our Conditional GAN and the WGAN-GP scheme for the library of vanilla GANs.



Experiment-IV: Device Modeling using GANs

■ Implementation

We use the MNIST dataset to validate our experiment setup.

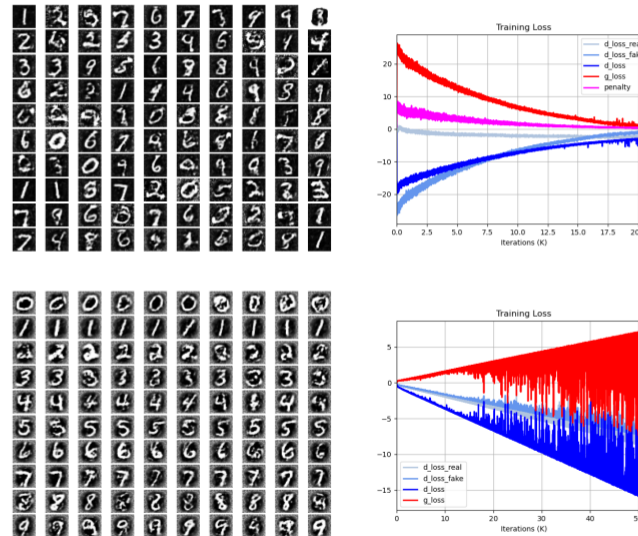
■ Accuracy

Each model is used to generate 50 predictions for each class. The Q-values for these predictions were computed and compared to the criterion for that class. It is observed that the accuracies of the generated designs for classes 0 and 1 are 62% and 80% in the case of the library of vanilla WGANs and are 80% and 98% in the case of the Conditional WGAN.

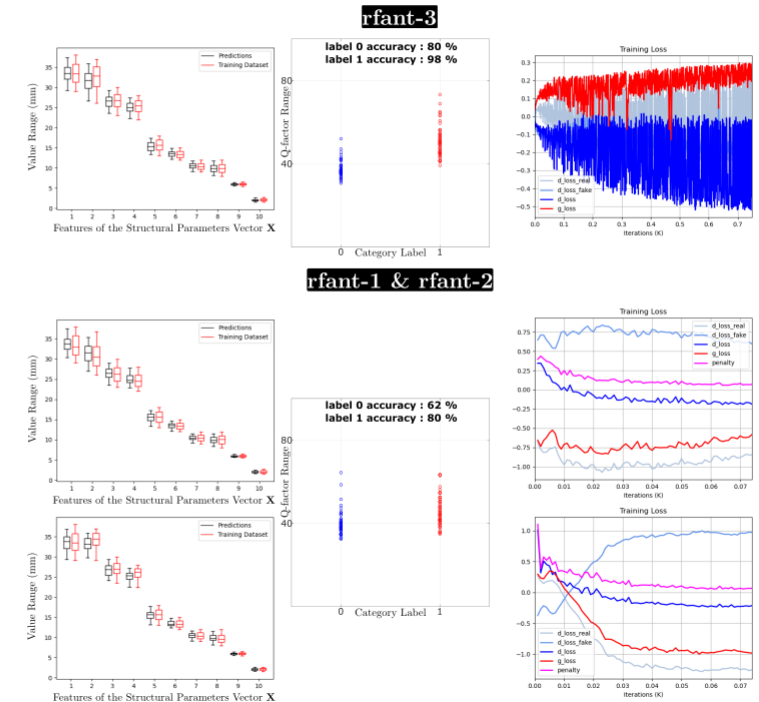
■ Latency

The average latency for predictions is 0.4 ms.

Validation of Experiment Setup



Synthetic Data Generation for rfant-x



repo: gitlab.com/oameed/unm_rfant_dloptant

Future Work Open Discussion

Publications/Talks

- Noakoasteen, O., et al. "**Physics-informed surrogates for electromagnetic dynamics using Transformers and graph neural networks**," in IET Microwaves, Antennas & Propagation, 2024. doi: [10.1049/mia2.12463](https://doi.org/10.1049/mia2.12463).
- O. Noakoasteen, C. Christodoulou, S. Goudos, "**Time-Domain Electromagnetic Analysis using Graph Neural Networks**," presented at the APS/URSI 2022: IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting. [Online] Paper [TU-A1.2P.1](#).
- O. Noakoasteen, J. Vijayamohanan, A. Gupta and C. Christodoulou, "**Antenna Design Using a GAN-Based Synthetic Data Generation Approach**," in IEEE Open Journal of Antennas and Propagation, vol. 3, pp. 488-494, 2022, doi: [10.1109/OJAP.2022.3170798](https://doi.org/10.1109/OJAP.2022.3170798).
- O. Noakoasteen, J. Vijayamohanan, A. Gupta, C. Christodoulou, "**Deep Surrogate Models for Time-Domain Electromagnetic Analysis using Attention: Going Beyond Recurrent Neural Networks**," presented at the APS/URSI 2021: IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting. [Online] Paper [TH-SP.2A.5](#).
- O. Noakoasteen, S. Wang, Z. Peng and C. Christodoulou, "**Physics-Informed Deep Neural Networks for Transient Electromagnetic Analysis**," in IEEE Open Journal of Antennas and Propagation, vol. 1, pp. 404-412, 2020, doi: [10.1109/OJAP.2020.3013830](https://doi.org/10.1109/OJAP.2020.3013830).
- J. Vijayamohanan, O. Noakoasteen, A. Gupta, M. Martínez-Ramón and C. G. Christodoulou, "**On Antenna Q-factor Characterization with Generative Adversarial Networks**," 2020 IEEE International Symposium on Antennas and Propagation and North American Radio Science Meeting, 2020, pp. 1643-1644, doi: [10.1109/IEEECONF35879.2020.9329612](https://doi.org/10.1109/IEEECONF35879.2020.9329612).
- O. Noakoasteen, S. Wang, Z. Peng, "**Deep Neural Network Representations of Transient Electrodynamical Phenomena**," presented at the APS/URSI 2019: IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting. [Online] Paper [MO-SP.1P.7](#), [Online] Abstract from [usnc-ursi-archive](#).
- O. Noakoasteen, S. Wang, Z. Peng, "**Physics-Informed Deep Neural Networks for Transient Electromagnetic Analysis**," presented at the 2019 IEEE MTT-S International Conference on Numerical Electromagnetic and Multiphysics Modeling and Optimization ([NEMO](#)).
- O. Noakoasteen, S. Wang, Z. Peng, "**Emulation of Transient Electrodynamical Physics via Deep Neural Networks**," presented at the APS/URSI 2018: IEEE International Symposium on Antennas and Propagation and USNC-URSI Radio Science Meeting. [Online] Paper TU-UB.1P.8, [Online] Abstract from [usnc-ursi-archive](#).